Unicenter

TCPaccess Communications Server Assembler API Programmer Reference

Version 6.0

Computer Associates

The Software That Manages eBusiness

Booklist

This documentation and related computer software program (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by Computer Associates International, Inc. ("CA") at any time.

This documentation may not be copied, transferred, reproduced, disclosed or duplicated, in whole or in part, without the prior written consent of CA. This documentation is proprietary information of CA and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, licensed users may print a reasonable number of copies of this documentation for their own internal use, provided that all CA copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the confidentiality provisions of the license for the software are permitted to have access to such copies.

This right to print copies is limited to the period during which the license for the product remains in full force and effect. Should the license terminate for any reason, it shall be the user's responsibility to return to CA the reproduced copies or to certify to CA that same have been destroyed.

To the extent permitted by applicable law, CA provides this documentation "as is" without warranty of any kind, including without limitation, any implied warranties of merchantability, fitness for a particular purpose or noninfringement. In no event will CA be liable to the end user or any third party for any loss or damage, direct or indirect, from the use of this documentation, including without limitation, lost profits, business interruption, goodwill, or lost data, even if CA is expressly advised of such loss or damage.

The use of any product referenced in this documentation and this documentation is governed by the end user's applicable license agreement.

The manufacturer of this documentation is Computer Associates International, Inc.

Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227-7013(c)(1)(ii) or applicable successor provisions.

© 2002 Computer Associates International, Inc. (CA)

All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Contents

Chapter 1: Assembler Language Macro Instructions

Chapter 2: DNR Directory Services

Appendix A: MF Operand Summary

Appendix B: Macro Instruction Operand Summary

Appendix C: Register Usage Summary

Appendix D: Data Structures (Assembler Language)

Index

Chapter 1

Assembler Language Macro Instructions

This chapter provides detailed coding information for the TCPaccess Communications Server API assembler macro instructions.

The following topics are covered in this chapter:

- Conventions, Definitions, and Terminology—Discusses the conventions and terminology used to describe the macro instructions, and defines the basic forms and formats that apply, and defines register usage for the standard API calling sequences.
- Macro Instructions Descriptions—Provides detailed descriptions of all API macro instructions.

Conventions, Definitions, and Terminology

Discusses the conventions and terminology used to describe the macro instructions, and defines the basic forms and formats that apply, and defines register usage for the standard API calling sequences.

Basic Format of Macro Descriptions

Each macro instruction is described using the following documentation conventions:

- A brief introductory statement summarizing macro instruction's primary use
- A detailed assembler format description
- A detailed description of each macro instruction operand
- A detailed assembler format description, and a detailed description of each macro instruction operand, including:
  - Completion information
  - A list of return codes presented in tabular format
  - General usage guidelines

The basic components of each macro instruction description are described in the following topics.

Assembler Format Description

The assembler format description is a three-column table that shows how the macro instruction should be coded. Because macro instructions are coded in the same format as assembler instructions, the three columns correspond to an assembler instruction's name, operation, and operand fields.

This is how these fields are used:

Name	The macro instruction name provides a label for the macro instruction. The name, if used, can be specified as any symbolic name valid in the assembler language.
Operation	This field contains the mnemonic operation code of the macro instruction. It is always coded exactly as shown.
Operands	The operands provide information required for macro instruction execution. Generally, this information is organized into a parameter list by macro instruction expansion, and provided to the API during program execution. All of the macro instruction's operands are shown in the operands column of the assembler format description. The notation used for describing macro instruction operands is similar to that used by IBM to describe macro instructions incorporated within their products (for example, VTAM macro instructions or MVS supervisor macro instructions). This notation is briefly defined in Macro Instruction Operand Notation .

Syntax Description

The syntax includes the assembler format, operand name, and description. Every operand description starts with a brief explanation of the operand's function. If more than one fixed valued can be supplied with the operand, the operand description also explains the effect that each value has on the action performed by the macro instruction.

For TPL-based (Transport Service Parameter List) macro instructions, the same operand can be coded on many different macro instructions. In some cases, the coding rules and interpretation of the operand are identical. In other cases, the coding rules may differ and the interpretation may depend on the context in which it is used.

Note: To avoid unnecessary repetition, the operands that apply to several macro instructions are documented in full only for the TPL macro instruction. For other macro instructions, either an abbreviated description is given, or only the characteristics that apply to the particular macro instruction are described.

The operand description may include a description of the format in which the operand should be coded. This description is provided when the format is an exception to these general rules:

- When a quantity is indicated, such as a length or integer value, it can be coded as any non-relocatable expression that is valid for an A-type address constant or a Load Address (LA) assembler instruction, or the number of the register (enclosed in parentheses) that contains the value when the macro instruction is executed. Register notation is restricted to registers 2-12 when specifying a quantity.
- When an address is indicated, it can be coded as any relocatable expression that is valid for an A-type address constant or a Load Address (LA) assembler instruction, or the number of the register (enclosed in parentheses) that contains the address when the macro instruction is executed. Register notation is restricted to registers 1-12 when specifying the address of a TPL, and registers 2-12 for all other address operands.

Note: The assembler instructions generated to process operands in the macro instruction expansion vary depending on the macro form used. This can have an effect on the maximum values that can be specified in certain macro instruction operands. For more information, refer to the discussion of list, generate, modify and execute forms of macro instructions in List, Generate, Modify, and Execute Forms .

Completion Information

All of the executable macro instructions pass return codes in registers, and most indicate status information in various control block fields when they are posted complete. Some macro instructions also return results in control block fields or storage areas provided by the application program. A description of the status information and data returned by the macro instruction follows the operand descriptions. This description is in terms of the conditions that exist on return to the application program after execution of the macro instruction in synchronous mode, or after execution of a TCHECK macro instruction when operating in asynchronous mode.

Return Codes

A list of all return code values generated by the macro instruction is presented in tabular form. The list is organized to show the conditional completion codes returned for normally completing macro instructions, and the combinations of recovery action codes and specific error codes returned for abnormally completing macro instructions.

All return codes are listed using their symbolic name. The TCPaccess *Unprefixed Messages and Codes* manual contains detailed information on each return code, including its symbolic name, the corresponding decimal and hexadecimal values, and a detailed explanation of its use and meaning.

Usage Information

The usage information describes the macro instruction's function and use, including any important rules that can apply to when and how it may be used. Special usage notes that may not have been covered in other sections of the macro instruction description are listed here.

Macro Instruction Operand Notation

The notational scheme used in the operands column of the assembler format description is similar to that used by IBM. This notation shows how, when, and where operands can be coded. These notational symbols are never coded in the macro instruction:

- Uppercase characters must be coded exactly as shown in the operands column. Lowercase italicized characters or words represent variables, or values that the application programmer must provide.
- A vertical bar (|) is the symbol for *exclusive or*.

  SYNC | ASYNC means that either SYNC or ASYNC (but not both) should be coded.
- An underscored value means that if the operand is omitted, the macro instruction will be expanded as though the underscored value had been coded.

  Note: The underscored value is the default value.

  In this example, INTERNAL is the default value. If the ECB operand is omitted, the assembler assumes ECB=INTERNAL.

  `ECB=INTERNAL | event_control_block_addr`

  Default values apply *only* to declarative macro instructions and to the list (MF=L) or generate (MF=G) forms of TPL-based macro instructions.

For all other macro instructions, as well as the modify (MF=M) or execute (MF=E) forms of TPL-based macro instructions, no default values are assumed. In this case, only the values specified in the macro instruction itself or already contained in the referenced control blocks are used.

- Brackets ([ ]) denote optional operands or values. Conversely, the lack of brackets indicates that an operand is required.

  In this example, the positional operand VERBATIM | SUMMARY is optional, and the MF operand is mandatory:

  ```
  [VERBATIM | SUMMARY,] MF=(E,tpl_address)
  ```

- An ellipsis (...) indicates that whatever precedes it (either an operand value or an entire operand) can be repeated any number of times.

  An operand appearing as

  ```
  (data structure name,...)
  ```

  could be coded as (TPL,TIB) if two data structure names are required, or as (TPL) or TPL if only one data structure name is required.

- Parentheses, equal signs, and commas must be coded exactly as shown, subject to the previously described rules.

Macro Instruction Operand Types

All macro instruction operands are either keyword or positional operands. Most of the API macro instruction operands are keyword operands.

Keyword Operands

Keyword operands consist of:

- A fixed character string (the operand keyword)
- An equal sign (=)
- A single or multiple operand value

The presence of the equal sign distinguishes keywords from positional operands.

You do not need to code keyword operands in the order shown in the operands column of the format description.

Example

A macro instruction having these operands indicated in the operands column

```
[,DABUF=user_data_address]
[,DALEN=user_data_length]
```

could be coded as:

```
DABUF=BUFFER,DALEN=200
```

or

```
DALEN=200,DABUF=BUFFER
```

Keyword operands must be separated by commas. If a keyword operand is omitted, the commas that would have been included with it are also omitted.

There are a few instances in the API macro instructions where more than one value can be coded after the keyword. When this is done, operand sublist notation is used.

The option code operand specified as:

```
OPTCD= ([ SHORT | LONG ]
        [ ,SYNC | ASYNC ]
        [ ,NEGOT | NONEGOT ]
        [ ,UNCOND | COND ]
        [ ,DEBUG | NODEBUG ] )
```

could be coded as OPTCD=ASYNC or as OPTCD=(ASYNC) when only one option code is used.

When more than one option code is used, however, the option code names must be enclosed in parentheses and separated by commas, in this format:

```
OPTCD=(SHORT,ASYNC,NONEGOT,UNCOND,DEBUG)
```

If a field name is omitted, the comma that would have been included with it is also omitted.

Omitting the first, third, and fourth option code names in this example results in this format:

```
OPTCD=(ASYNC,DEBUG)
```

Positional Operands

Positional operands are used to a much lesser extent, and must be coded in the exact order shown in the operands column when more than one positional operand is shown.

Positional operands are separated by commas as are all operands, but if you omit a positional operand, the surrounding commas must still be coded to indicate its absence.

A macro instruction that has the three positional operands OPTIONS, ADDRESS, and LENGTH would be coded as:

```
OPTIONS,ADDRESS,LENGTH
```

or as if all three are coded:

```
,ADDRESS,LENGTH
```

if only the last two are coded.

Note: If the last positional operand or operands are omitted, the trailing comma or commas should not be coded.

Operand Coding Order

Positional operands are generally coded before any keyword operands, but this is not required as long as the previous rules are followed. You can code all operands on a single line, separate lines, or a combination of both. When coding operands on more than one line, the standard assembler language rules for continuation lines apply.

List, Generate, Modify, and Execute Forms

The standard form (that is, no MF operand indicated) of a nondeclarative API macro instruction expands into both nonexecutable code that represents the parameters specified on the macro instruction, and executable code that causes the API routines to be entered when the macro instruction is executed. The nonexecutable code, called the parameter list, is assembled at the point in the application program where the macro instruction appears. For TPL-based macro instructions, this parameter list is usually a short form TPL.

Standard Macro Instruction Disadvantages

Strict use of standard form macro instructions simplifies the design of an application program, but has these major disadvantages:

- Because the parameter list is expanded inline with executable code, the application is rendered nonreentrant
- Parameter lists for TPL-based macro instructions cannot be shared

Alternative API Macro Instructions

Alternative forms of the API macro instructions are provided that overcome one or both of the disadvantages described in the preceding section. These various forms cause the assembler to respond in one of these ways:

- Build the parameter list where the macro instruction appears in the application program, but assemble no executable code (nonreentrant list form)
- Assemble code that builds the parameter list at a location indicated at execution time, but assembles no executable code to cause the API routines to be entered (reentrant list form)
- Assemble code that builds the parameter list at a location indicated at execution time, and assembles code that causes the appropriate API routine to be entered to execute the requested function (generate form)
- Assemble code that modifies an existing parameter list at execution time, but assembles no executable code to cause the API routines to be entered (modify form)
- Assemble code that modifies an existing parameter list and causes the appropriate API routine to be entered to execute the requested function (execute form)

The following tables summarize the various macro instruction forms.

Actions Taken for Various Macro Instruction Forms

The following table lists the actions taken during assembly and execution for each form, and how it is specified on the macro instruction.

Form	Actions Taken:		Coded With
	During Assembly	During Execution	
Standard	Parameter list and code to execute function assembled where macro instruction appears in application program.	Inline parameter list modified and requested function executed.	No MF operand or MF=I
List (Nonreentrant)	Parameter list assembled where macro instruction appears in application program.	No executable code (execute form must be used).	MF=L
List (Reentrant)	Executable code assembled to build parameter list at location indicated by application program.	Parameter list built, but requested function not executed (execute form required).	MF=(L,address)
Generate	Code assembled to build parameter list and execute requested function.	Parameter list built at location indicated by application program and requested function executed.	MF=(G,address)
Modify	Code assembled to modify parameter list indicated by application program.	Existing parameter list modified but requested function not executed (execute form required).	MF=(M,address)
Execute	Code assembled to modify parameter list and execute requested function.	Existing parameter list modified and requested function executed.	MF=(E,address)

Note: The various alternative forms of the API macro instructions are designated with the MF operand.

Runtime Characteristics of Various Macro Instruction Forms

The following table lists the runtime characteristics of each form, whether defaults apply to unspecified operands, and how operands are generated by the assembled code.

Form	Runtime Characteristics:		Defaults Apply	Operand Type
	Reentrant	Shared TPL		
Standard	No	No	Yes	Address Constant
List (Non-reentrant)	No	Yes	Yes	Address Constant
List (Reentrant)	Yes	Yes	Yes	Register Displacement
Generate	Yes	No	Yes	Register Displacement
Modify	Yes	Yes	No	Register Displacement
Execute	Yes	Yes	No	Register Displacement

The operand types are the same as the ones used in the IBM Macro manual:

A Address Constant Type

RX Register Displacement Type

List Form

The MF operand for the list form of any non-declarative macro instruction is coded in this format:

```
MF = { L | ( L,address ) }
```

L Indicates that this is the list form of the macro instruction.

- If the format MF=L is coded, then the parameter list is assembled in place
- If the format MF=(L,address) is coded, then the parameter list is built during program execution at the specified address

When the list form is used, default values are generated for any unspecified macro instruction operands.

address Storage location where the parameter list is to be built during program execution.

This area must begin on a fullword boundary and if the application program is reentrant, must be in dynamically allocated storage. Because the assembler builds executable code that in turn builds the parameter list, the macro instruction must be in the executable portion of the application program.

`MF = ( G,`*`address`*` )`

G Generate form of the macro instruction.

Default values are generated for any unspecified macro instruction operands.

address Storage location where the parameter list will be built. Generally, this is in dynamically allocated storage.

This area must begin on a fullword boundary and if the application program is to be reentrant, must be in dynamically allocated storage. Because the assembler builds executable code that in turn builds the parameter list, the macro instruction must be in the executable portion of the application program. After the parameter list is built, the requested function executes.

Modify Form

The MF operand for the modify form of any non-declarative macro instruction is coded in this format:

`MF=(M,`*`address`*`)`

M Modify form of the macro instruction.

Default values are not generated for any unspecified macro instruction operands.

address Storage location of an existing parameter list that will be modified.

Code is assembled to modify those parameters corresponding to operands specified on the macro instruction. If an operand is not specified, the parameter is not modified and no default value is assumed.

The modify form of a macro instruction lets the application program modify a parameter list after it is built and before it is reused to execute another transport function. However, the parameter list cannot be expanded. If the parameter list is a short form TPL and the macro instruction attempts to modify a parameter beyond the range of the parameter list, execution of the macro instruction is terminated with a general return code of eight in register 15.

Execute Form

The MF operand for the execute form of any non-declarative macro instruction is coded in this format:

```
MF = ( E,address )
```

E — Execute form of the macro instruction.

Default values are not generated for any unspecified macro instruction operands.

address — Storage location of an existing parameter list to be modified. Code is assembled to modify those parameters corresponding to operands specified on the macro instruction.

If an operand is not specified, the parameter is not modified and no default value is assumed. After the parameter list is modified, the requested function executes.

For TPL-based macro instructions, the address indicated with the MF operand is the TPL itself. As an alternative to the execute form of the MF operand, TPL=address can be specified. In other words, TPL=address is equivalent to specifying MF=(E,*address*). This alternative is provided for VTAM programmers who may prefer the similarity to the RPL operand used in VTAM macro instructions.

Note: The TPL and MF operands must not be specified together.

Optional and Required Operands

The assembler format description shows most macro instruction operands as being optional. However, whether or not an operand is optional depends on the form of macro instruction used. Execute (MF=E) and modify (MF=M) forms update an existing parameter list, and therefore, all operands except the address of the parameter list itself are optional. The list forms (MF=L) assume that the parameter list may be updated later when it is executed with the execute form, and do not require all operands to be specified when the list is generated. Only the standard (MF=I) and generate (MF=G) forms require certain operands to be specified at assembly time, and even these are limited.

Since all API service functions (except TOPEN) require a valid endpoint identifier, the EP operand must be coded on the standard and generate forms of macro instructions.

The fact that an operand is optional for a given instance of a macro instruction does not mean the corresponding parameter is optional for the request. It is the responsibility of the application program to make sure that all required parameters are included in the parameter list when it is executed. The API does extensive checking of the parameter list, and terminates processing of a request if any required parameters are missing.

Default and Maximum Values

The default values assumed by a macro instruction for unspecified operands only apply to the standard, list, and generate forms. In this case, the macro instruction is expanded as if all unspecified operands were coded with their default values. For modify and execute forms, only operands that are specified are modified. If an operand is not specified, the value stored in the parameter list is used during the subsequent execution of the requested function.

The method used to store the operand value varies depending on the form used. When an inline parameter list is specified with the standard and nonreentrant list forms, A-type address constants are generally expanded which generate the operand values. After assembly, the operand values are contained at the proper locations within the parameter list.

For all other forms, and when register notation is not specified, a Load (L) or Load Address (LA) assembler instruction is usually expanded to load the value into register 14 or 15. The resulting value is stored at the proper location in the parameter list.

The use of a load address instruction has implications for the maximum value that can be specified using the latter forms. Address values are limited to the size of an address in the current addressing mode (24 or 31 bits), and quantity values are limited to 12 bits (4,095 decimal) unless an index or base register is specified, in which case it is limited to the size of an address. There is no restriction when register notation is used.

If LENGTH is an operand with an integer value, then the LENGTH=number argument is limited to a 12-bit value, the LENGTH=displacement(index,base) argument is limited to a 24-bit or 31-bit value, and the LENGTH=(register) argument can be a 32-bit value.

Rules for Loading Operand Values

Whether a Load (L) or Load Address (LA) instruction is used to load an operand value depends on the nature of the operand.

The following rules apply:

- If the operand is a simple integer value, or quantity (for example, the length or size of some object), it is loaded with a Load Address (LA) instruction.
- If the operand value is the address of some object, and the object itself can be generated at assembly time in static storage (for example, an ECB address, the address of an exit routine, or a storage area containing a protocol address), it is loaded with a Load Address (LA) instruction.
- All other operand values (for example, endpoint identifiers, TCB or ASCB addresses, and sequence numbers returned by the API) are loaded with a Load (L) instruction. These are generally values acquired from some other source.

The appendix, "Macro Instruction Operand Summary" lists the operand formats for all API macro instructions and the default values that apply when operand values are not specified.

Linkage Conventions

TPL-based macro instructions (standard, generate, and execute forms) all use the same conventions to call the API routines.

Register Contents on Routine Entry

On entry to the API routine, registers are set as shown in register contents on routine entry.

R0	Function code.
R1	Address of the parameter list (TPL).
R2-R12	Unmodified application program registers.
R13	Address of 18-word register save area.
R14	Address of the next sequential instruction in the application program.
R15	The API entry point address.

Register Contents on Return

On return from the API routine, the registers are set as shown in Register Contents on Return.

R0	Recovery action or conditional completion code.
R1	Address of parameter list (TPL).
R2 - R12	Unmodified application program registers.
R13	Address of 18-word register save area.
R14	Address of the next sequential instruction in the application program.
R15	General return code (zero if successful).

The function code passed in register zero can be negated to indicate some function-specific option. The values returned in register zero and 15 can be modified by the SYNAD or LERAD exit routine if the request completed abnormally. The appendix "Register Usage Summary" summarizes register usage conventions employed by the API.

Macro Instructions Descriptions

This section provides detailed descriptions of all API macro instructions, arranged in alphabetical order.

Each macro instruction description includes this information:

- The name of the macro instruction
- A brief statement of its function and use
- The assembler format description
- A detailed description of each operand
- A description of completion information returned
- A table of return codes
- General usage information

It is assumed that you are familiar with the API concepts and facilities presented in *TCPaccess Assembler API Concepts*.

ACLOSE

Terminate Session with the API Subsystem— The ACLOSE macro instruction is used to terminate a session between the application program and the API. The APCB used to establish the session is the sole operand of the ACLOSE macro instruction.

```
[ symbol ] ACLOSE APCB_address
```

APCB_address Address of an opened APCB (Application Program Control Block) that defines the session with the API.

Unless supplied in a general register, the address of the APCB is loaded into register one using a Load Address (LA) instruction. If register notation is specified, the address of the APCB can be contained in any one of the general registers 1-12.

Only one APCB can be closed with the ACLOSE macro instruction. An invalid or corrupted value causes unpredictable results.

Default: None (must be specified).

Completion Information

After the ACLOSE macro instruction completes successfully, the APCB is closed and the session established when the APCB was opened is terminated. Fields modified by the API when the APCB was opened are returned to their original values, and any resources allocated by the API on behalf of the transport user are released. In particular, any transport endpoints associated with the APCB are closed. If ACLOSE completes unsuccessfully, the session is not terminated.

When control is returned to the next sequential instruction following the ACLOSE macro instruction, successful completion is indicated by a return code of zero in register 15, and an error code of zero in register zero. The error code field in the APCB is also set to zero. If the ACLOSE macro instruction is unsuccessful, an error code is returned in register zero, and in most cases, is also stored in the APCB.

Unsuccessful completion is indicated by one of these nonzero return codes.

4 (X'04') The APCB was already closed.

The error code in register zero is set to APCBECLS, and the APCB is unmodified.

12 (X'0C') The ACLOSE macro instruction failed and the error is permanent.

The error code returned in register zero is also stored in the APCB, and indicates the reason for the failure. The permanent error flag is not set in the APCB.

16 (X'10') A fatal error occurred and the APCB could not be closed.

An error code is returned in register zero, but the error code field in the APCB is not changed. This return code is generally the result of an invalid APCB.

The value returned in register zero indicates the nature of the error encountered by the ACLOSE macro instruction.

Return Codes

If the APCB was determined to be valid, was not already closed and was not busy, the API also returns this value in the APCBERRC field of the APCB. An APCB is considered busy if it is in the process of being opened or closed by another task. The following table lists these error codes, which are defined in more detail in the TCPaccess *Unprefixed Messages and Codes* manual.

Return Code	(Register 15)	Specific Error Code (Register 0)		
0	(X'00')	0		
4	(X'04')	APCBECLS		
12	(X'0C')	APCBEPRB APCBETRV	APCBELER APCBEENV	APCBEAMD APCBEEND
16	(X'10')	APCBELER	APCBEVCK	APCBEBSY

Usage Information

The ACLOSE macro instruction closes (or *deactivates*) an APCB and terminates the session established when the APCB was opened.

The ACLOSE macro instruction:

- Closes any endpoints opened by the transport user
- Releases all resources associated with the APCB
- Prevents the opening of any more endpoints by the transport user

Note: Fields within the APCB that were filled in by the API when the APCB was opened are reset to their original value, and the APCB should not be referenced unless reopened by another AOPEN macro instruction.

The ACLOSE macro instruction must be issued in the mainline program. That is, the ACLOSE macro instruction must be executed from a PRB, and no IRBs or SVRBs can exist in the current RB chain. The addressing mode in effect must also be consistent with the APCB when it was opened. If the APCB was opened in 31-bit mode, and RMODE=ANY was coded on the APCB (the default), the APCB must also be closed in 31-bit mode. There is only one form of the ACLOSE macro instruction, and its expansion is always reentrant. If the task that opened an APCB terminates before closing the APCB, an ACLOSE is issued by the API task termination exit.

AOPEN

Establish Session with the API Subsystem — The AOPEN macro instruction establishes a session between the application program and the API. Parameters describing the application program, and specifying the access method to use, are provided in an APCB.

The address of the APCB is the sole operand of the AOPEN macro instruction.

```
[ symbol ] AOPEN APCB_address
```

APCB_address

The APCB to associate with the transport user issuing the AOPEN macro instruction. Unless supplied in a general register, the address of the APCB is loaded into register one using a Load Address (LA) instruction.

If register notation is specified, the address of the APCB can be contained in any one of the general registers 1-12. Only one APCB can be opened with the AOPEN macro instruction.

Default: None (must be specified).

Completion Information

After the AOPEN macro instruction completes successfully, the APCB is initialized and a session is established with the API. Certain fields within the APCB are modified during AOPEN processing to reference data areas used by the API. The application program should not modify any information within the APCB while it is opened. These fields are restored to their original values when the APCB is closed.

When control is returned to the next sequential instruction following the AOPEN macro instruction, successful completion is indicated by a return code of zero in register 15, and an error code of zero in register zero. The error code field in the APCB is also set to zero.

If the AOPEN macro instruction completes unsuccessfully, an error code is returned in register zero, and in most cases, is also stored in the APCB. Unsuccessful completion is indicated by one of these nonzero return codes.

4 (X'04')	The APCB was already opened. The error code in register zero is set to APCBEOPN, and the APCB is not modified.
8 (X'08')	The AOPEN macro instruction failed because of some temporary condition. The error code returned in register zero is also stored in the APCB, and indicates the reason for the failure. Retry the AOPEN macro instruction later.
12 (X'0C')	The AOPEN macro instruction failed and the error is permanent. The error code returned in register zero is stored in the APCB and indicates the reason for the failure. The permanent error flag is also set in the APCB, and must be cleared before you can use the APCB with another AOPEN macro instruction.
16 (X'10')	A fatal error occurred and the APCB could not be opened. An error code is returned in register zero, but the error code field in the APCB is unchanged. This return code is generally the result of an invalid APCB.

The value returned in register zero indicates the nature of the error encountered by the AOPEN macro instruction. If the APCB is valid, not already opened, and not busy or flagged with a permanent error, the API returns this value in the APCBERRC field of the APCB.

The following table lists these error codes. They are defined in more detail in the TCPaccess *Unprefixed Messages and Codes*.

Return Code	(Register 15)	Specific Error Code (Register Zero)		
0	(X'00')	0		
4	(X'04')	APCBEOPN		
8	(X'08')	APCBERDY	APCBECVT	
12	(X'0C')	APCBEPRB APCBECFG APCBESTP APCBEOPT	APCBELER APCBEVCK APCBEDRA APCBEENV	APCBEACT APCBEBEG APCBEVER APCBEDUP
16	(X'10')	APCBELER APCBEBSY	APCBEVCK	APCBEPER

Usage Information

The AOPEN macro instruction opens (or *activates*) an APCB, and establishes a session between the application program and the API. The APCB defines a specific transport user (that is, the issuing task) and is used to associate subsequent service requests with this transport user. The APCB also serves as an anchor for the API data structures and transfer vectors required for subsequent requests execution.

When the AOPEN macro instruction completes successfully, fields within the APCB contain information stored by the API. This information is used to process future service requests and should not be modified by the application program while the APCB is open. When the APCB is closed, these fields are returned to their original, pre-opened condition. Modification of any of these fields while the APCB is open causes unpredictable results.

The API runs as a separate job in its own address space. Because an application program can be started before the API, an application program may issue an AOPEN macro instruction before the API is active. In this case, the AOPEN fails, and the application program is informed that the API is inactive.

The following possibilities exist, each of which is indicated by a separate error code:

- The API subsystem was not configured in the MVS operating system, because either it is not yet installed, or it has not been activated since the last IPL.
- The API subsystem is configured, but is not active. This is considered a permanent error, and the application programmer should check with the system operator to determine why the API is not active.
- The API subsystem is active, but has not completed initialization, and is not ready to service requests. This is considered a temporary error; the AOPEN macro instruction can be retried after some delay.
- The API subsystem is ready, but the access method interface has not completed initialization. This error is similar to the previous error and should be retried after some delay.

  The addressing mode in effect at the time the AOPEN macro instruction is executed determines the residency mode of data areas allocated by the API. The location of the APCB must also be consistent with the current addressing mode.
- If the addressing mode is 24-bit, all dynamic storage is allocated with LOC=BELOW, and the APCB must reside below 16 MB. Future service requests can then be executed in any addressing mode.
- If the addressing mode is 31-bit, all storage is allocated with LOC=ANY, and all future service requests must be issued in 31-bit addressing mode. The APCB can reside above or below 16 MB.

Note: The RMODE parameter on the APCB can be used to force allocation below 16 MB, thus allowing the APCB to be opened in 31-bit mode, and future requests to be issued in 24-bit mode.

The AOPEN macro instruction must be issued in the mainline program (that is, the AOPEN macro instruction must be executed from a PRB, and no IRBs or SVRBs can exist in the current RB chain).

There is no limit on the:

- Number of operand APCBs for an address
- Number of tasks per address space that can have currently-opened APCBs

Each APCB defines a different transport user.

There is only one form of the AOPEN macro instruction, and its expansion is always reentrant. However, since AOPEN modifies the APCB, the APCB may need to be moved to dynamically allocated storage before AOPEN is executed if the application program is to be reentrant.

APCB

Generate an Application Program Control Block— The APCB macro instruction is used to generate an Application Program Control Block (APCB). The APCB identifies a transport user, and contains information that is used to service requests issued by the transport user. The address of the APCB is supplied as an operand of an AOPEN and ACLOSE macro instruction. The address of an opened APCB must also be provided whenever a new transport endpoint is created.

```
[ symbol ] APCB [ AM = TLI ]
                [ ,EXLST = exit_list_address ]
                [ ,APPLID = application_name ]
                [ ,PASSWD =application_password ]
                [ ,SYSID = TCPaccess_API_subsys_name ]
                [ ,ENVIRO = ASM | IBMC | SASC ]
                [ ,ACNTX = application_level context ]
                [ ,ECNTX =environment_level_context ]
                [ ,OPTCD = ( [ TRACE | NOTRACE ]
                             [ ,AUTHEXIT | NOAUTHEXIT ] ) ]
                [ ,RMODE=24 | ANY ]
                [ ,MF = L | DSECT ]
```

AM = TLI

Access method used by the application program. TLI must be coded as shown when using the transport layer interface.

Default: TLI (Transport Layer Interface).

EXLST = *exit_list_address*

Address of an exit list to associate with the APCB. The exit list contains addresses of user exit routines to be entered when certain events occur. The exit list can be generated by the TEXLST macro instruction, and should specify AOPEN, indicating that the exit list will be linked to the transport user via AOPEN.

An exit list identified by the APCB applies to all endpoints opened by the transport user. You can provide a separate exit list for each endpoint by identifying the exit list with TOPEN. The same exit list can be referenced by more than one APCB.

If no exit list is specified, the application program cannot receive asynchronous notification when the corresponding events occur. In this case, the application program must process such events synchronously by analyzing the return code at the completion of each service function.

Default: Zero (no APCB exit list).

APPLID = *application_name*

Name of the application program. The name must be coded as an alphanumeric string up to eight characters in length. If the name is longer than eight characters, it is truncated to eight bytes. The application name is used in combination with the password to authorize access to the API services. If a user ID is not provided when endpoints are opened, the application name is also used to authorize endpoint services.

If no name is provided, a null string of eight zero bytes is generated. If this field is still set to zero when the APCB is opened, the API substitutes the step name from the TIOT. This is either the procedure step name if the job step was invoked via a procedure or the job step name otherwise.

Default: Null name (use TIOT step name).

PASSWD = *application_password*

Password associated with the application name. The password may be any alphanumeric string up to eight characters in length. If the password is longer than eight characters, it is truncated to eight bytes. If the password operand is not coded, a null password is generated consisting of eight zero bytes.

Default: Null password (no password provided).

SYSID = TCPaccess_*API_subsys_name*

The TCPaccess API runs as an MVS subsystem, and is initially located via its subsystem name when the APCB is opened with an AOPEN macro instruction. This subsystem name is specified when the API is installed. If the subsystem name defined during installation does not agree with the default name used by the APCB macro instruction or if more than one instance of the API can run on the local system, you can specify the subsystem name using this operand.

The subsystem name is an alphanumeric string up to four characters in length. If the subsystem name you specify is longer than four characters, it is truncated to four bytes. If no subsystem name is coded, the default name is used.

Default: ACSS.

ENVIRO = ASM | IBMC | SASC

Runtime environment for the application program. Generally, the runtime environment is assembler language, and ASM should be coded.

IBMC and SASC are reserved for the API library routines that execute in the runtime environment of the IBM and SAS/C compilers, respectively. This operand selects special interface exits that initialize and terminate the runtime environment, and that schedule user exit routines written in a higher-level language.

Normal application programs should either leave this operand unspecified, or always specify ENVIRO=ASM. Specifying other environments yields unpredictable, and unsatisfactory, results.

Default: ASM (assembler language environment).

ACNTX=
application_level_context

An arbitrary fullword of user context associated with the application program. This information is not interpreted by the API, and is saved in the APCB, and included in the parameter list provided to exit routines.

This information can be used by the application program to derive application-level context associated with the transport user during exit processing. Any value that can be generated as an A-type address constant can be specified.

Note: The value stored in the APCB is moved to another data area after the APCB is opened. If the application program changes the value in the APCB after AOPEN is executed, it is not reflected in the value passed to the exit routine.

Default: Zero (no application-level context specified).

ECNTX =
environment_level_context

An arbitrary fullword of user context associated with the runtime environment of the application program.

This information is not interpreted by the API, and is saved in the APCB, and included in the parameter list provided to exit routines. Any value that can be generated as an A-type address constant can be specified.

Note: The value stored in the APCB is moved to another data area after the APCB is opened. If the application program changes the value in the APCB after AOPEN is executed, it is not reflected in the value passed to the exit routine.

This information is intended for use by built-in interface routines that map the assembler language runtime environment of the API into the runtime environment of the application program.

If ENVIRO=ASM is coded, this field can be used for any purpose by the application program.

If ENVIRO=ASM is not coded, this field is reserved for use by the runtime environment exits.

Default: Zero (no environment-level context specified).

OPTCD = TRACE | NOTRACE

Indicates whether service requests issued by the transport user associated with this APCB are traced by the API.

OPTCD=TRACE events associated with endpoints linked to this APCB are traced.

If OPTCD=TRACE is selected when the APCB is opened, a storage area is allocated within the application program's address space for tracing endpoint events. All events associated with a given transport user are co-mingled in time-order sequence.

The following events are traced:

- Transport service function invoked
- Service request rejected
- WAIT SVC issued by TCHECK
- Service request completed
- Asynchronous completion exit entered
- Asynchronous event exit entered
- Synchronous error exit entered
- TCHECK control function completed
- TERROR control function completed
- TSTATE control function completed

The occurrence of an event is recorded by an eight-word trace entry stored in a circular trace table. The information saved in each trace entry depends on the type of event.

Exit events save the first six words of the TXP associated with the exit. All other events save the first five words of the TPL associated with the event, and its address. All events contain a time-stamp generated by a store clock (STCK) instruction.

OPTCD=NOTRACE tracing is disabled for this transport user.

Default: TRACE (trace endpoint events).

OPTCD = AUTHEXIT | NOAUTHEXIT

Indicates if exits associated with endpoints or with the API session are to be driven in fast-path, or authorized, mode.

OPTCD=AUTHEXIT exits are given control in SRB mode, key zero, and supervisor state.

OPTCD=NOAUTHEXIT exits are driven in IRB or PRB mode, caller key, and problem state.

Default: NOAUTHEXIT.

RMODE = 24 | ANY

Residency mode of any storage allocated in the application program's address space by the API.

RMODE=24 storage is always allocated below 16 MB. Local data areas used by the API are always allocated below 16 MB and are always addressable no matter what addressing mode is in effect.

RMODE=ANY storage is allocation in accordance with the current addressing mode. Storage is allocated above 16 MB if the APCB is opened in 31-bit mode and below 16 MB otherwise. Therefore, the API service functions associated with an APCB opened in 31-bit mode and RMODE=ANY must be executed in 31-bit mode.

The addressing mode of all subsequent service requests associated with this APCB must be consistent with the addressing mode in effect when the APCB is opened, as well as the RMODE parameter of the APCB itself.

The application program is also responsible for assuring that the residency mode of data areas it manages (for example, the APCB and TPLs) is compatible with the addressing mode. The API performs consistency checks on the addressing mode whenever a service request is issued. However, it is possible that unpredictable results can occur before the API has had an opportunity to perform this check.

Default: ANY (allocate storage above 16 MB).

MF = L | DSECT

Macro expansion type.

MF=L APCB is generated inline with the APCB macro instruction.

MF=DSECT A DSECT that maps the fields of the APCB is generated.

There is no remote list form of the APCB macro instruction. If the application program is reentrant and must generate an APCB in dynamic storage, it should allocate the storage area and move a copy of the APCB into it. The skeleton APCB can be generated in static storage as long as it is not opened with the AOPEN macro instruction.

Default: MF=L (inline list).

Completion Information

The APCB macro instruction is declarative and generates no executable instructions. Refer to the description of the AOPEN and ACLOSE macro instructions for completion information.

Return Codes

The APCB contains fields in which return codes are stored during AOPEN and ACLOSE processing. If either of these macro instructions complete with an error, error information generally is returned in these fields. APCBERRC contains a one-byte specific error code, and APCBDGNC contains a two-byte diagnostic code. The codes that can be returned in these fields are defined in the TCPaccess *Unprefixed Messages and Codes* manual.

Usage Information

Each application program must define a transport user before it can get the services of a transport provider. A transport user is defined by creating an APCB containing information required by the API and then activating the APCB by referencing it in an AOPEN macro instruction. The task that issues the AOPEN macro instruction becomes permanently associated with the APCB and is considered the transport user.

Opening the APCB causes the API to create the necessary infrastructure for servicing the transport user, and pointers to various components of the infrastructure are stored in the APCB. This information must not be modified while the APCB is opened. The APCB serves as an anchor for all information associated with the transport user and the address of this APCB must be provided each time the transport user opens a new endpoint.

An application program can have more than one transport user, and each transport user must be associated with a unique APCB. If two instances of the API happen to be running on the local system (each with a different subsystem name), and an APCB is opened by the same task for both, the transport user defined by one APCB is completely independent of the other.

The APCB can also be thought of as defining a session between the application program and the API. The AOPEN macro instruction establishes the session, and the ACLOSE macro instruction terminates the session. Once a session is established, the transport user can request services such as opening endpoints, binding protocol addresses, and transferring data. When the APCB is closed, the session is terminated and such requests can no longer be made. Any endpoints opened by the transport user are closed and all resources allocated to the transport user are released.

The APCB serves as an anchor for the API data areas and contains information about the transport user. An exit list of routines to enter when certain events occur is also linked to the APCB. An arbitrary word of user context stored in the APCB is provided with each entry to any of these exit routines. This information is not examined or interpreted by the API.

The APCB can reside in 24-bit or 31-bit storage. The application program is required to assure that the location of the APCB and any data areas it references is consistent with current addressing mode. In particular, neither the executable instructions expanded by API macro instructions nor the interface routines they call, change the current addressing mode, and operate in the addressing mode of the caller. It is generally advisable that all future requests made to the API be issued in the same addressing mode in effect when the APCB is opened.

TACCEPT

Accept a Connection Request— When a connect indication is received at an endpoint with a TLISTEN macro instruction, the TACCEPT macro instruction is used to accept the connection request, and to establish a connection with the remote transport user. The connection can be established to a new endpoint, or to the endpoint on which the TLISTEN was executed.

```
[ symbol ] TACCEPT [ EP = endpoint_id ]
                   [ ,NEWEP = new_endpoint_id ]
                   [ ,SEQNO =sequence_number ]
                   [ ,OPTCD = ( [ SHORT | LONG | EXTEND]
                   [ ,SYNC | ASYNC ] ) ]
                   [ ,ECB = INTERNAL | event_control_block_addr ]
                   [ ,EXIT = tpl_exit_routine_address ]
                   [ ,MF = ( I | L | G | M | E ,[ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TACCEPT macro instruction is to be executed.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

NEWEP = *new_endpoint_id*

Endpoint at which the accepted connection is established.

The operand value is the endpoint identifier returned by a TOPEN macro. A zero value indicates the listening endpoint (that is, the endpoint at which the connect indication arrived).

The application program can accept a connection on the listening endpoint or a newly created endpoint. If accepted on the listening endpoint, the endpoint must not have any pending connect indications other than the one being accepted. The endpoint must be in the connect-indication-pending state (TSINCONN), and have only one indication pending in its queue. Otherwise, the TACCEPT macro instruction is completed abnormally.

If the connection is accepted on a new endpoint, the endpoint must be in the disabled state (TSDSABLD), and must have been opened by the task that opened the listening endpoint. The local protocol address bound to the endpoint can be the same address bound to the listening endpoint, or different. Connecting to an endpoint with a different protocol address may not be supported by all transport providers.

Default: Zero (connection established to listening endpoint) .

SEQNO = *sequence_number*

Connect indication to accept. The value specified must be returned by a TLISTEN macro instruction. The transport provider uses this value to identify a connect indication pending for this endpoint which has not yet been accepted or rejected.

Default: Zero (most likely an invalid sequence number).

OPTCD = SHORT | LONG | EXTEND

Format attribute of the parameter list associated with this request.

OPTCD=SHORT a different control block identifier is used to indicate that a subset of the TPL has been generated.

OPTCD=LONG a standard, full-length TPL is generated.

OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters.

Default: SHORT if MF=I or MF operand omitted, LONG otherwise.

OPTCD = SYNC | ASYNC	Indicates the synchronization mode to use when executing the TACCEPT macro instruction. OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction is complete. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API. OPTCD=ASYNC the request executes in asynchronous mode, and returns control immediately after scheduling the TACCEPT request. The application program is responsible for issuing the TCHECK macro instruction. Default: SYNC (synchronous mode).
ECB = INTERNAL | *event_control_block_addr*	Location of an Event Control Block (ECB) to be posted by the API when the TACCEPT macro instruction associated with this TPL has completed. The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB. The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB. This operand is mutually exclusive with the following EXIT operand. Default: INTERNAL (internal ECB).
EXIT = *tpl_exit_routine_address*	Address of an exit routine to schedule when the TACCEPT macro instruction associated with this TPL has completed. The TPL exit routine is scheduled only if asynchronous mode was specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB. This operand is mutually exclusive with the previous ECB operand. Default: Not indicated (no TPL exit routine).

MF = (I | L | G | M | E, [*tpl_address*])

Standard, list, generate, modify, or execute form of the TACCEPT macro instruction.

The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used.

See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: MF=I (standard, nonreentrant form).

Completion Information

The TACCEPT macro instruction completes normally (or conditionally) once the accepted connection is established.

- If the connection is accepted to itself, the state of the endpoint is changed from connect-indication-pending (TSINCONN) to connected (TSCONNCT), and the endpoint is ready to send and receive data.
- If the connection is accepted to a new endpoint, the state of the new endpoint is changed from disabled (TSDSABLD) to connected (TSCONNCT), and the old endpoint can continue receiving connect indications.
- If only one connect indication was pending, the state of the old endpoint is changed from connect-indication-pending (TSINCONN) to enable (TSENABLD). Otherwise, the state of the endpoint is unchanged.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY), and a conditional completion code is returned in register zero. The TPL return code field is set accordingly. No other information is returned.

If the TACCEPT macro instruction completes abnormally, no connection is established, and the connect indication remains queued. If the connection request was abandoned via a disconnect, the indication is removed from the queue when the TCLEAR macro instruction executes. The state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TACCEPT return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY		
TRFAILED	TAEXCPTN	TENONEGO		
	TAINTEG	TEPROTO	TEDISCON	
	TAENVIRO	TESYSERR TESTOP TEUNSUPF	TESUBSYS TETERM TEUNAUTH	TEDRAIN TEUNSUPO TERSOURC
	TAFORMAT	TEBDFNCD TEBDEXIT TEBDSQNO	TEBDOPCD TEBDDATA TEBDEPID	TEBDECB TEBDOPTN
	TAPROCED	TEAMODE TENOCONN TEOWNER	TESTATE TEINDICA	TEINCMPL TEACCEPT
	TATPLERR	TEACTIVE		
TRFATLFC	func. code	The function code loaded into register zero is invalid.		
TRFATLPL	diag. code	The TPL address is invalid, or the TPL has been corrupted.		
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.		
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.		

Usage Information

The TACCEPT macro accepts a pending connect indication that was received at an endpoint. A sequence number returned by a previous TLISTEN macro instruction identifies which connect indication will be accepted in the event more than one is pending. On the successful completion of this request, the specified endpoint is connected, and is ready to send and receive data.

The connection can be established to the endpoint at which the connect indication was received, or established to a new endpoint. If established to the endpoint that issued the corresponding TLISTEN macro instruction, no more than one connect indication can be pending. The maximum number of pending connect indications is limited to the queue length specified when the endpoint was enabled (see TBIND). Once the connection is established, the listening endpoint becomes busy, and does not generate any additional connect indications. However, connection requests arriving at the endpoint may continue to be queued by the transport provider.

Application programs that establish a connection to the listening endpoint operate as single-threaded servers. Multithreaded servers, on the other hand, must leave the endpoint available to receive additional connect indications. In this case, a new endpoint must be created for each connection that is established. The endpoint must exist in the same communications domain, and use the same type of service as the listening endpoint. The local protocol address bound to the new endpoint is generally the same as the listening endpoint.

The endpoint identifier provided with the TACCEPT macro instruction identifies the endpoint to which the connection is established. If the value specified for NEWEP is zero, the connection is established to the listening endpoint (if possible). If the endpoint ID identifies a different endpoint, the endpoint must have been opened by the same task that opened the listening endpoint, and must be in the disabled (TSDSABLD) state.

The TACCEPT macro instruction is normally used to establish connections to endpoints operating in connection mode. However, if the endpoint is operating in connectionless mode, and was enabled to simulate connect indications, the TACCEPT macro instruction can be used to create an association with a transport user whose protocol address was returned with the TLISTEN macro instruction. See *Assembler API Concepts* for a discussion on associations in connectionless mode.

TADDR

Retrieve Local or Remote Protocol Address— The local protocol address bound to an endpoint, or the remote protocol address of a connected (or associated) transport user, can be retrieved using the TADDR macro instruction.

```
[ symbol ] TADDR [ EP = endpoint_id ]
                 [ ,ADLEN = protocol_address_length ]
                 [ ,ADBUF = protocol_address_address ]
                 [ ,ADALET = protocol_address_alet ]
                 [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                              [ ,SYNC | ASYNC ]
                              [ ,TRUNC | NOTRUNC ]
                              [ ,LOCAL | REMOTE ] ) ]
                 [ ,ECB = INTERNAL | event_control_block_addr ]
                 [ ,EXIT = tpl_exit_routine_address ]
                 [ ,MF = ( I | L | G | M | E, [ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TADDR macro instruction executes.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

ADLEN = *protocol_address_length*

Length (in bytes) of the protocol address storage area identified by the ADBUF operand.

The length is updated when the request is completed to reflect the actual length of the protocol address returned. If the length is zero, the TADDR macro instruction is abnormally completed.

Default: Zero (return no protocol address).

ADBUF = *protocol_address_address*

Address of a storage area for returning the protocol address of the designated transport user.

The storage area should be large enough to contain the entire address. The format of the protocol address is provider-dependent, and its maximum size can be determined by issuing a TINFO macro instruction. The storage area can be aligned on any boundary.

Default: Zero (no protocol address storage area).

ADALET = *protocol_address_alet*	Access List Entry Token (ALET) used in access register (AR) mode when referencing the storage specified by the ADBUF parameter. The ADALET value must be an ALET contained in the Dispatchable Unit Access List (DUAL) of the caller. The ADALET parameter can be used only if OPTCD=EXTEND is also specified. Default: Zero (the storage is contained in the address space of the caller).
OPTCD = SHORT | LONG | EXTEND	Format attribute of the parameter list associated with this request. OPTCD=SHORT a different control block identifier is used to indicate that a subset of the TPL has been generated. OPTCD=LONG a standard, full-length TPL is generated. OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters. Default: SHORT if MF=I or MF operand omitted, LONG otherwise.
OPTCD = SYNC | ASYNC	Synchronization mode to use when executing the TADDR macro instruction. OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction is complete. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API. OPTCD=ASYNC the request executes in asynchronous mode, and returns control immediately after scheduling the TADDR request. The application program is responsible for issuing the TCHECK macro instruction. Default: SYNC (synchronous mode).
OPTCD = TRUNC | NOTRUNC	Indicates whether the protocol address returned to the application program by the transport provider should be truncated if it does not fit within the storage area provided. OPTCD=TRUNC the excess is truncated, and the TADDR macro instruction is completed conditionally as long as no other errors occur.

	OPTCD=NOTRUNC nothing is placed in the storage area, and the TADDR macro instruction is completed abnormally. Default: NOTRUNC (no truncation).
OPTCD = LOCAL | REMOTE	Protocol address to return to the application program. OPTCD=LOCAL the protocol address of the local transport user, which was bound to the endpoint by the application program, is returned. OPTCD=REMOTE the protocol address of the remote transport user connected to, or associated with, the endpoint is returned. Default: LOCAL (local protocol address).
ECB = INTERNAL | *event_control_block_addr*	Location of an Event Control Block (ECB) to be posted by the API when the TADDR macro instruction associated with this TPL completes. The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB. The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB. This operand is mutually exclusive with the following EXIT operand. Default: INTERNAL (internal ECB).
EXIT = *tpl_exit_routine_address*	Address of an exit routine to schedule when the TADDR macro instruction associated with this TPL completes. The TPL exit routine is scheduled only if asynchronous mode is specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB. This operand is mutually exclusive with the previous ECB operand. Default: Not indicated (no TPL exit routine).

MF = (I | L | G | M | E, [*tpl_address*])

Standard, list, generate, modify, or execute form of the TADDR macro instruction.

The second sublist operand, tpl_address, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used.

See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: MF=I (standard, nonreentrant form).

Completion Information

The TADDR macro instruction completes normally (or conditionally) when the designated protocol address is returned to the application program. The length of the storage area is updated to reflect the actual length of the protocol address. The negotiated length of the connect indication queue is also returned in the QLSTN field of the TPL associated with this request. The state of the endpoint is not changed.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY), and a conditional completion code is returned in register zero. TCTRUNC is set if the protocol address returned to the application program was truncated to fit in the storage area provided. The TPL return code field is set accordingly. No other information is returned.

If the TADDR macro instruction completes abnormally, no information is returned. The state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TADDR return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY	TCTRUNC	
TRFAILED	TAINTEG	TEOVRFLO		
	TAENVIRO	TESYSERR TESTOP	TESUBSYS TETERM	TEDRAIN
	TAFORMAT	TEBDFNCD TEBDEXIT	TEBDOPCD TEBDADDR	TEBDECB
	TAPROCED	TEAMODE	TESTATE	TEINCMPL
	TATPLERR	TEACTIVE		
RFATLFC	func. code	The function code loaded into register zero is invalid.		
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.		
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.		
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.		

Usage Information

The TADDR macro instruction is used to retrieve the protocol address of the local or remote transport user associated with an endpoint. The local protocol address can be retrieved any time after a TBIND macro instruction has been successfully executed at the endpoint. The remote protocol address can be retrieved any time after a TACCEPT or TCONFIRM macro instruction is successfully executed, completing the connection or association with a remote transport user.

Generally, this information is returned to the application program by the macro instruction that bound the local protocol address, or completed the connection or association. However, if the application program did not provide a storage area in which the protocol address could be returned, or needs to acquire the information again, the TADDR macro instruction can be used. An example of the latter is after an endpoint that has already been bound to a local protocol address, is passed to another task or address space. The negotiated length of the connect indication queue can also be acquired in this manner.

If the TADDR macro instruction is executed at an endpoint operating in connectionless mode, either the bound local protocol address, or the remote protocol address associated with the last received datagram, is returned as indicated by the OPTCD operand.

When a local protocol address is retrieved, a partial protocol address may be returned (that is, a portion of the protocol address may be unknown at the time the TADDR macro instruction was issued, and the corresponding field is set to zero). In particular, if a TADDR macro instruction is issued before a COTS endpoint is connected, the network address portion of the protocol address may be set to zero. This is because multi-homed hosts (that is, hosts with more than one network connection) cannot determine the local network address until the destination is known. In the case of a CLTS endpoint, the local network address may change if datagrams are transmitted or received via different network connections.

TBIND

Bind a Protocol Address to a Transport Endpoint— The TBIND macro instruction is used to bind a local protocol address to an endpoint, and to define the number of connections that can be pending for the endpoint. When completed, a COTS endpoint is ready to begin connection establishment, and a CLTS endpoint is ready to begin data transfer.

```
[ symbol ] TBIND [ EP=endpoint_id ]
                 [ ,ADLEN = protocol_address_length ]
                 [ ,ADBUF = protocol_address_address ]
                 [ ,ADALET = protocol_address_alet ]
                 [ ,QLSTN = listen_queue_length ]
                 [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                              [ ,SYNC | ASYNC ]
                              [ ,TRUNC | NOTRUNC ]
                              [ ,NEGOT | NONEGOT ]
                              [ ,ASSIGN | USE ] ) ]
                 [ ,ECB = INTERNAL | event_control_block_addr ]
                 [ ,EXIT = tpl_exit_routine_address ]
                 [ ,MF = ( I | L | G | M | E , [ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TBIND macro instruction executes.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

ADLEN = *protocol_address_length*

Length (in bytes) of the protocol address, or protocol address storage area, identified by the ADBUF operand.

If the storage area is for retrieving a protocol address from the transport provider, the length indicated is the maximum length of the storage area. Otherwise, this operand indicates the length of the protocol address contained in the storage area. The length may be zero, indicating there is no protocol address or storage area.

Default: Zero (no protocol address).

ADBUF = *protocol_address_address*

Address of a protocol address storage area that contains a local protocol address assigned by the application program or a local protocol address assigned and returned by the transport provider.

Option codes specified with the OPTCD operand determine how the storage area is used. If the local protocol address is assigned by the transport provider, the storage area must be large enough to contain the returned address. The format and maximum size of a protocol address is provider-dependent, and can be determined by issuing a TINFO macro. The storage area can be aligned on any boundary.

Default: Zero (no protocol address storage area).

ADALET = *protocol_address_alet*

Access List Entry Token (ALET) used in access register (AR) mode when referencing the storage specified by the ADBUF parameter.

The ADALET value must be an ALET contained in the Dispatchable Unit Access List (DUAL) of the caller. You can only us the ADALET parameter if OPTCD=EXTEND is also specified.

Default: Zero (the storage is contained in the address space of the caller).

QLSTN = *listen_queue_length*

Size of the queue for holding incoming connections arriving at the endpoint, and pending connect indications received by the application program.

If the value specified is zero, no connections can be queued, and the endpoint is disabled for receiving connects.

If the value specified is nonzero, incoming connections are queued, and corresponding connect indications are generated at the endpoint.

A connect indication remains pending until it is accepted or rejected by the application program, or until the connection is abandoned by the caller. The value of this operand generally determines whether the application program is operating in client or server mode.
The transport provider may not be able to queue the number of connections specified by the application program, and as a result, attempts to negotiate the indicated value to a lesser amount. If negotiation is permitted by the application program .

	(OPTCD=NEGOT), the request completes conditionally, and returns a conditional completion code in register zero. Otherwise, the TBIND request completes abnormally. Default: Zero (endpoint is disabled).
OPTCD = SHORT | LONG | EXTEND	Format attribute of the parameter list associated with this request. OPTCD=SHORT a different control block identifier is used to indicate that a subset of the TPL was generated. OPTCD=LONG a standard, full-length TPL is generated. OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters. Default: SHORT if MF=I or MF operand omitted, LONG otherwise.
OPTCD = SYNC | ASYNC	Synchronization mode to use when executing the TBIND macro instruction. OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction is complete. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API. OPTCD=ASYNC the request executes in asynchronous mode, and returns control immediately after scheduling the TBIND request. The application program is responsible for issuing the TCHECK macro instruction. Default: SYNC (synchronous mode).

OPTCD = TRUNC | NOTRUNC	Indicates whether the protocol address returned to the application program by the transport provider should be truncated if it does not fit within the storage area provided. OPTCD=TRUNC the excess is truncated and the TBIND macro instruction is completed conditionally as long as no other errors occur. OPTCD=NOTRUNC nothing is placed in the storage area and the TBIND macro instruction is completed abnormally. Default: NOTRUNC (no truncation).
OPTCD = NEGOT | NONEGOT	Indicates whether the value specified for QLSTN can be negotiated to a lesser value. OPTCD=NEGOT (and the value is larger than can be accommodated by the transport provider), a smaller value is used. The negotiated value is returned to the application program, and the TBIND function completes conditionally as long as no other errors occur. OPTCD=NONEGOT no negotiation is performed and the TBIND function completes abnormally. Default: NoneGOT (negotiation disallowed).
OPTCD = ASSIGN | USE	Indicates whether the application program or the transport OPTCD=ASSIGN, the transport provider assigns an appropriate address. If ADLEN and ADBUF designate a storage area, the transport provider returns the protocol address assigned. OPTCD=USE the transport provider uses the protocol address provided by the application program. The storage area designated by the ADLEN and ADBUF operands must contain a valid protocol address. Default: USE (use protocol address provided by application program).

ECB = INTERNAL | *event_control_block_addr*	Location of an Event Control Block (ECB) to be posted by the API when the TBIND macro instruction associated with this TPL completes. The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB. This operand is mutually exclusive with the following EXIT operand. Default: INTERNAL (internal ECB).
EXIT = *tpl_exit_routine_address*	Address of an exit routine to schedule when the TBIND macro instruction associated with this TPL completes. The TPL exit routine is scheduled only if asynchronous mode was specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB. This operand is mutually exclusive with the previous ECB operand. Default: Not indicated (no TPL exit routine).
MF = (I | L | G | M | E , [*tpl_address*])	Standard, list, generate, modify, or execute form of the TBIND macro instruction. The second sublist operand, tpl_address, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters. Default: MF=I (standard, nonreentrant form).

Completion Information

The TBIND function completes normally (or conditionally) when the local protocol address is bound to the specified endpoint. The state of the endpoint is changed to disabled (TSDSABLD) or enabled (TSENABLD) in accordance with the value specified for QLSTN. Any protocol address assigned by the transport provider is returned in the storage area provided by the application program. The length of the storage area is updated to reflect the actual length of the assigned protocol address.

- If the endpoint is operating in connectionless mode, then it is ready to send and receive datagrams
- If the endpoint is operating in connection mode and disabled, the endpoint can be used to initiate a connection request
- If the endpoint is operating in connection mode and is enabled, the endpoint can receive connect indications generated by arriving connection requests

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY), and a conditional completion code is returned in register zero. TCNEGOT is set in the conditional completion code if the value specified for QLSTN was negotiated to a lesser value. The negotiated value is returned in the same TPL storage location. TCTRUNC is set if the storage area was too small to contain the entire protocol address returned by the transport provider. The TPL return code field is set accordingly. No other information is returned.

If the TBIND function completes abnormally, the state of the endpoint is unchanged. If the state of the endpoint was opened (TSOPENED), no protocol address is bound to the endpoint. If the state was disabled (TSDSABLD), the endpoint remains in its disabled condition, and the length of the connect indication queue is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TBIND return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY	TCTRUNC	TCNEGOT
TRFAILED	TAEXCPTN	TENONEGO		
	TAENVIRO	TESYSERR TESTOP TERSOURC	TESUBSYS TETERM TEINUSE	TEDRAIN TEUNAUTH
	TAFORMAT	TEBDFNCD TEBDEXIT	TEBDOPCD TEBDADDR	TEBDECB TEBDQLEN
	TAPROCED	TEAMODE	TESTATE	TEINCMPL
	TATPLERR	TEACTIVE		
TRFATLFC	func. code	The function code loaded into register zero is invalid.		
TRFATLPL	diag. code	The TPL address is invalid, or the TPL has been corrupted.		
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.		
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.		

Usage Information

The TBIND macro instruction associates a local protocol address with a transport endpoint. Once a protocol address is bound, an application program may begin sending and receiving datagrams through the endpoint if operating in connectionless mode, or may request a connection to a remote transport user if operating in connection mode. Optionally, a connection-mode endpoint can be enabled for receiving connection requests from a remote transport user.

The local protocol address can be assigned by the application program or the transport provider.

- If assigned by the application program (OPTCD=USE), then the protocol address is placed in a storage area designated by the ADLEN and ADBUF operands, and is passed to the transport provider for binding to the endpoint
- If assigned by the transport provider (OPTCD=ASSIGN), then the storage area is used to return the assigned address

Generally, an application operating in server mode assigns the protocol address, and an application operating in client mode defers this responsibility to the transport provider.

The application program can only assign the local transport address; the local network address is always assigned by the transport provider. Therefore, when assigning a protocol address, the application program should set the network address field to zero, or provide only a partial (that is, truncated) protocol address. Similarly, the protocol address returned by the API contains a null network address. The local network address is not assigned until a connection is established, or a datagram is received at the endpoint. If the application program requires the local network address, it should execute a TADDR macro instruction after the connection is established at a COTS endpoint, or after a datagram has been transmitted or received at a CLTS endpoint.

Note: The local network address bound to a connectionless-mode endpoint may change with each datagram transmitted or received via the endpoint.

Application programs operating in client mode are active, and as soon as a local protocol address is bound, the endpoint can be used to initiate a connection to the server. Application programs operating in server mode are passive, and wait for the client to send a connection request. Arriving connections are queued by the transport provider, and cause connect indications to be generated at the endpoint. The number of connections that can be queued at the endpoint, and the number of connect indications that can be pending, is determined by the value of QLSTN specified when the local protocol address was bound.

Connections that arrive at the endpoint are queued until accepted or rejected by the application program. When the queue is full, subsequent connection requests are discarded. A larger value for QLSTN reduces the chances of a transport user finding the endpoint busy, and gives the application more time to respond to a connect indication. A value of zero disables the endpoint from receiving any connect indications. Application programs operating in client mode must leave the endpoint disabled by setting QLSTN to zero.

Note: The value of QLSTN does not limit the number of transport users that can be simultaneously connected to the application program. The number of pending connections is limited by the QLSTN value; the number of established connections is limited by the number of requests the application program is willing to accept. One endpoint must be created for each accepted connection (see TACCEPT).

The value specified for QLSTN by application programs operating in connectionless mode should normally be zero. However, if a nonzero value is specified, the transport provider simulates connection-mode service (that is, the first arriving datagram generates a connect indication, which the application program is expected to receive via a TLISTEN macro instruction, and accept or reject with a TACCEPT or TREJECT macro instruction). If accepted, an association is created for the endpoint, and TRECV and TSEND macro instructions can be used to send and receive data. If the server is multithreaded, datagrams arriving from other transport users generate new connect indications.

The binding of the local protocol address and enabling of the endpoint can be separated into two distinct requests issued to the transport provider. If the first request binds a protocol address and leaves the endpoint disabled by specifying a QLSTN value of zero, a second TBIND macro instruction can be executed to enable the endpoint. If the second instance of TBIND specifies a protocol address, it must be the address already bound to the endpoint.

The format and content of a protocol address is provider-dependent. To minimize dependence on a particular provider, or particular protocol, applications operating in server mode should use directory services to map service names into local protocol addresses. Clients should let the transport provider assign the protocol address, and should use directory services to get addresses of remote hosts and services. The chapter “DNR Directory Services” provides information on using Network Directory Services (NDS) supported by Unicenter TCPaccess.

TCHECK

Check Status of a Transport Service Request— The TCHECK macro instruction is used to check the completion status of an active TPL. If necessary, a system WAIT macro instruction is issued to wait for completion, and if the request completed abnormally, the SYNAD or LERAD exit routine is entered.

```
[ symbol ] TCHECK MF = ( E, tpl_address )
```

MF = (E, *tpl_address*)

Execute form of the TCHECK macro instruction.

The second sublist operand, *tpl_address*, specifies the address of the TPL associated with the request whose completion status is being checked.

See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: None (must be coded as indicated).

Completion Information

The TCHECK macro instruction completes normally or conditionally when the request associated with the TPL has been posted complete, and the TPL has been set inactive. The state of the endpoint is updated to reflect the actions taken by the API routines. If an internal or external ECB is used for synchronization, the ECB is cleared.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY) to indicate successful completion, and the conditional completion code (if any) is returned in register zero. The TPL return code field is set accordingly. Other information may be returned in the TPL or storage areas provided by the application program in accordance with the particular service requested by the TPL.

If the TCHECK macro instruction completes abnormally, the TPL associated with the request is set inactive, the internal or external ECB (if any) is cleared, and the state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero indicates the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The return codes passed in registers zero and 15, and stored in the TPL return code field, are set in accordance with the particular request associated with the TPL. The return codes that are valid for each macro instruction are listed with the description of the macro instruction. The return codes listed in the following table are those that can be generated during TCHECK processing:

Note: If the SYNAD or LERAD exit routine is scheduled, the values in register zero and 15 at the next sequential instruction following the macro instruction are the values returned by the exit routine.

The following table lists the symbolic names for the TCHECK return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation
TRFAILED	TAPROCED	TEINACTV
	TATPLERR	TEB4EXIT
TRFATLFC	func. code	The function code loaded into register zero is invalid.
TRFATLPL	diag. code	The TPL address is invalid, or the TPL has been corrupted.
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.

Usage Information

The TCHECK macro instruction is used to check the completion status of a TPL-based request executed in asynchronous mode. If the request is incomplete, the task issuing the TCHECK macro instruction is suspended until posted complete. The TPL is set inactive so it can be used with another request, and if the request completed abnormally, the SYNAD or LERAD exit routine (if any) is entered.

When a TPL-based request is executed in asynchronous mode (that is, OPTCD=ASYNC was indicated by macro instruction), the API returns to the application program immediately after scheduling the requested function. If the return code in register 15 is zero (TROKAY), the request is accepted and executed asynchronously. The application program must subsequently issue a TCHECK macro instruction to check for completion of the request.

Request completion is indicated in one of these ways:

- Posting of an ECB
- Entering a TPL exit routine

The mechanism used for a particular request is indicated by the ECB or EXIT macro instruction operands, and the actions performed by the TCHECK macro instruction depend on which was indicated.

When the TCHECK macro instruction is executed for a TPL-based request that specified an ECB, these actions occur:

- If the request being checked has not been completed, execution of the application program task that issued the TCHECK macro instruction is suspended until the request is completed. However, any exit routines associated with the endpoint can still run.
- The ECB (internal or external) is cleared.

Note: The ECB specified in the TPL-based macro instruction must not be cleared between the time the macro instruction is issued and the corresponding TCHECK macro instruction is issued. If the ECB is cleared during this interval, the application program task may be suspended indefinitely.

- The TPL being checked is marked inactive and available for reuse by another TPL-based macro instruction.
- If the request being checked completed normally or conditionally (as indicated by the return code), control is returned to the next sequential instruction following the TCHECK macro instruction.

- If the request being checked completed abnormally, the SYNAD or LERAD exit routine is invoked, if available. Otherwise, control is returned to the next sequential instruction following the TCHECK macro instruction.
- If the SYNAD or LERAD exit routine is invoked, and the exit routine returns to the address in register 14 at the time it was entered, control is returned to the next sequential instruction following the TCHECK macro instruction. In this case, the contents of register zero and 15 are those returned by the exit routine.

When a TPL exit routine is used in place of an ECB, the TCHECK macro instruction should be issued within the exit routine. These actions occur:

- The TPL being checked is marked inactive and available for reuse by another TPL-based macro instruction.
- If the request being checked completed normally or conditionally (as indicated by the return code), control is returned to the next sequential instruction following the TCHECK macro instruction.
- If the request being checked completed abnormally, the SYNAD or LERAD exit routine is invoked, if available. Otherwise, control is returned to the next sequential instruction following the TCHECK macro instruction.
- If the SYNAD or LERAD exit routine is invoked, and the exit routine returns to the address in register 14 at the time it was entered, control is returned to the next sequential instruction following the TCHECK macro instruction. In this case, the contents of register zero and 15 are those returned by the exit routine.
- If issued outside the exit routine, and before the exit routine is invoked, the TCHECK macro instruction completes abnormally.

A TCHECK request can only be issued against a TPL that is marked active. Abnormal completion occurs if issued for an inactive TPL that is inactive. The TCHECK macro instruction should never be issued for a TPL associated with a synchronous request. When operating in synchronous mode, the API performs TCHECK equivalent processing automatically, and processing proceeds as though an internal ECB was specified.

TCLEAR

Acknowledge (Clear) Disconnect Indication—The TCLEAR macro instruction is issued to clear a disconnect indication pending at an endpoint, and to receive the disconnect reason code and any data accompanying the disconnect.

```
[ symbol ] TCLEAR [ EP = endpoint_id ]
                  [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                               [ ,SYNC | ASYNC ] ) ]
                  [ ,ECB = INTERNAL | event_control_block_addr ]
                  [ ,EXIT = tpl_exit_routine_address ]
                  [ ,MF = ( I | L | G | M | E , [ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TCLEAR macro instruction is to be executed.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

OPTCD = SHORT | LONG | EXTEND

Format attribute of the parameter list associated with this request.

OPTCD=SHORT a different control block identifier is used to indicate that a subset of the TPL was generated.

OPTCD=LONG a standard, full-length TPL is generated.

OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters.

Default: SHORT if MF=I or MF operand omitted, LONG otherwise.

OPTCD = SYNC | ASYNC

Synchronization mode to use when executing the TCLEAR macro instruction.

OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction is complete. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API.

OPTCD=ASYNC the request executes in asynchronous mode, and returns control immediately after scheduling the TCLEAR request. The application program is responsible for issuing the TCHECK macro instruction.

Default: SYNC (synchronous mode).

ECB = INTERNAL | *event_control_block_addr*	Location of an Event Control Block (ECB) to be posted by the API when the TCLEAR macro instruction associated with this TPL is completed. The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB. The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB. This operand is mutually exclusive with the following EXIT operand. Default: INTERNAL (internal ECB).
EXIT = *tpl_exit_routine_address*	Address of an exit routine to be scheduled when the TCLEAR macro instruction associated with this TPL is completed. The TPL exit routine is scheduled only if asynchronous mode has been specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB. This operand is mutually exclusive with the previous ECB operand. Default: Not indicated (no TPL exit routine).
MF = (I | L | G | M | E , [*tpl_address*])	Standard, list, generate, modify, or execute form of the TCLEAR macro instruction. The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used. See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters. Default: MF=I (standard, nonreentrant form).

Completion Information

The TCLEAR macro instruction completes normally (or conditionally) when the pending disconnect indication is cleared, and any disconnect user data received with the disconnect is moved into the storage area provided by the application program. The length of the storage area is updated to reflect the actual amount of disconnect user data returned.

The state of the endpoint is changed in accordance with the value specified for QLSTN by a TBIND macro instruction. If the endpoint cannot receive connect indications, the state is changed to disabled (TSDSABLD). Otherwise, the state is changed to enabled (TSENABLD) unless other connect indications are pending, in which case the state is unchanged (that is, the current state must have been connect-indication-pending (TSINCONN).

A protocol-dependent reason code is returned in the DISCD field of the TPL which specifies the reason for the disconnect. If the disconnect was received for a pending connect indication that is being abandoned by the remote transport user, the associated sequence number is returned in the TPLSEQNO field of the TPL.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY), and a conditional completion code is returned in register zero. TCTRUNC is set if the disconnect user data returned to the application program was truncated to fit in the storage area provided. The TPL return code field is set accordingly. No other information is returned.

If the TCLEAR macro instruction completes abnormally, no information is returned, and the disconnect indication (if any) remains queued. The state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TCLEAR return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY	TCTRUNC	
	TAINTEG	TEPROTO	TEOVRFLO	
	TAENVIRO	TESYSERR TESTOP TEUNSUPF	TESUBSYS TETERM	TEDRAIN TEUNSUPO
	TAFORMAT	TEBDFNCD TEBDEXIT	TEBDOPCD TEBDDATA	TEBDECB
	TAPROCED	TEAMODE TENODISC	TESTATE	TEINCMPL
	TATPLERR	TEACTIVE		
TRFATLFC	func. code	The function code loaded into register zero is invalid.		
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.		
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.		
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.		

Usage Information

The TCLEAR macro instruction is used to clear a pending disconnect indication, and to receive any user data sent with the disconnect request. This macro instruction should only be issued if a disconnect indication exists at the endpoint.

A disconnect indication is generated when a disconnect request is received at an endpoint, and can occur under any one of the following conditions:

- A connection request initiated by a TCONNECT macro instruction is pending, and the called transport user rejected the request
- A connection request initiated by a TCONNECT macro instruction was accepted by the called transport user, but before the application issued a TCONFIRM macro instruction completing the connection, the transport user disconnected
- A connect indication that was received with a TLISTEN macro instruction is pending, and the calling transport user disconnected before it was accepted with a TACCEPT macro instruction, or rejected with a TREJECT macro instruction
- A connection was fully established, and the peer transport user disconnected while data transfer was in progress
- Data transfer was complete, but before the application program could disconnect, the peer transport user requests a disconnect
- The application program initiated an orderly release with the TRELEASE macro instruction, and the peer transport user disconnected before the complementary orderly release was received with the TRELACK macro instruction
- The peer transport user initiated an orderly release that was received by a TRELACK macro instruction, but disconnected before the application program completed the orderly release with a TRELEASE macro instruction

These disconnect indications are all initiated as the result of some action by the remote transport user. The transport provider may also generate a disconnect indication on its own. This typically occurs when the transport provider fails to establish or maintain a connection because of a protocol error or network malfunction.

A disconnect causes immediate release of a connection. If the connection was fully established, any data buffered at the endpoint is discarded, and no additional data can be sent. The only information queued at the endpoint is the disconnect user data that was sent with the disconnect request. Once a disconnect indication is pending, only a TCLEAR (or TCLOSE) macro instruction should be issued.

A limited amount of user data may be received with the disconnect indication if supported by the transport provider. The content of this data is application-dependent, and not interpreted by the API or the transport provider. The maximum length of user data that can be received can be determined by issuing a TINFO macro instruction. Disconnects initiated by the transport provider are never accompanied by user data.

A disconnect reason code is returned by the transport provider. The value and interpretation of the reason code is provider-dependent. These are some possible reasons:

- Invoked by remote transport user (disconnect user data may contain additional information)
- Invoked by the transport provider due to the lack of resources
- Quality of service below minimum level
- Invoked by the transport provider due to protocol error or fatal malfunction
- Called transport user is unknown or unreachable
- Called transport user is unavailable
- Invoked by transport provider, but no reason given

If a disconnect occurs on an enabled endpoint while a connect indication is pending, a disconnect indication is generated. The indication can be presented synchronously with the completion of a TACCEPT (or TREJECT) macro instruction, or generated asynchronously by scheduling the DISCONN exit (see TEXLST). In either case, when the TCLEAR macro instruction is executed, a sequence number is returned. The sequence number should be used by the application program to determine which connection request to abandon.

The TCLEAR macro instruction is normally only executed at endpoints operating in connection mode. However, if an association has been established for an endpoint operating in connectionless mode, the transport provider may generate a disconnect indication under certain abnormal conditions. In this case, the indication must be cleared with a TCLEAR macro instruction, which also serves to terminate the association.

TCLOSE

Close a Transport Endpoint— The TCLOSE macro instruction closes an endpoint, and alternatively, to pass control of the endpoint to another task or address space. If the endpoint is to be deleted, any resources associated with the endpoint are released.

```
[ symbol ] TCLOSE [ EP = endpoint_id ]
                  [ ,TCB = task_control_block_address ]
                  [ ,ASCB = address_space_control_block_address / ANY]
                  [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                               [ ,SYNC | ASYNC ]
                               [ ,DELETE | PASS ] ) ]
                  [ ,ECB = INTERNAL | event_control_block_addr ]
                  [ ,EXIT = tpl_exit_routine_address ]
                  [ ,MF = ( I | L | G | M | E, [ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TCLOSE macro instruction is to be executed.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

TCB = *task_control_block_address*

TCB address of a task that acquires control of the endpoint when OPTCD=PASS is specified.

If the indicated value is zero, control is passed to the first task in the specified address space, which issues a complementary TOPEN macro instruction.

If the indicated value is not zero, control can only be passed to the indicated task.

Default: Zero (pass to any task).

ASCB = *address_space_control_block_address* | ANY

ASCB address of another address space that acquires control of the endpoint when OPTCD=PASS is specified.

If the indicated value is zero, the endpoint can only be passed to a task executing in the same address space.

If the indicated value is not zero, the indicated value is the ASCB address of another address space containing the task that acquires control.

If the indicated value is ANY, the endpoint can be passed to any address space in the system.

Default: Zero (pass to task in this address space).

OPTCD = SHORT | LONG | EXTEND	Format attribute of the parameter list associated with this request. OPTCD=SHORT a different control block identifier is used to indicate that a subset of the TPL has been generated. OPTCD=LONG a standard, full-length TPL is generated. OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters. Default: SHORT if MF=I or MF operand omitted, LONG otherwise.
OPTCD = SYNC | ASYNC	Synchronization mode to use when executing the TCLOSE macro instruction. OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction is complete. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API. OPTCD=ASYNC the request executes in asynchronous mode, and returns control immediately after scheduling the TCLOSE request. The application program is responsible for issuing the TCHECK macro instruction to complete the TCLOSE request. Default: SYNC (synchronous mode).
OPTCD = DELETE | PASS	Disposition of the endpoint designated in the TPL associated with this macro instruction. OPTCD=DELETE the endpoint is closed, and any record of the endpoint is deleted from all internal tables and local storage. OPTCD=PASS the endpoint is not closed, and control of the endpoint is passed to the designated task or address space that issued the corresponding TOPEN (OPTCD=OLD). When control is being passed, the TCLOSE request does not complete until a complementary TOPEN (OPTCD=OLD) macro instruction is issued by the acquiring task or address space. Default: DELETE (delete endpoint).

ECB = INTERNAL | *event_control_block_addr*	Location of an Event Control Block (ECB) to be posted by the API when the TCLOSE macro instruction associated with this TPL is completed. The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB. The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB. This operand is mutually exclusive with the following EXIT operand. Default: INTERNAL (internal ECB).
EXIT = *tpl_exit_routine_address*	Address of an exit routine to be scheduled when the TCLOSE macro instruction associated with this TPL is completed. The TPL exit routine is scheduled only if asynchronous mode has been specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB. This operand is mutually exclusive with the previous ECB operand. Default: Not indicated (no TPL exit routine).
MF = (I | L | G | M | E , [*tpl_address*])	Standard, list, generate, modify, or execute form of the TCLOSE macro instruction. The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used. See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters. Default: MF=I (standard, nonreentrant form).

Completion Information

The TCLOSE macro instruction completes normally when the endpoint is closed, or control has been passed to the acquiring task.

- If the OPTCD=DELETE operand was indicated, the state of the endpoint is changed to closed (TSCLOSED)
- If the OPTCD=PASS operand was indicated, the state of the endpoint is unchanged

In this case, the TCLOSE macro instruction does not complete until the acquiring task successfully executes a matching TOPEN macro instruction indicating OPTCD=OLD, at which time it becomes the controlling task.

The TCB and ASCB addresses of the acquiring task are returned in the TPL.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY). The conditional completion code in register zero is always zero (TCOKAY), and the TPL return code field is set accordingly. No other information is returned.

If the TCLOSE macro instruction completes abnormally, the issuing task continues to control the endpoint. The state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TCLOSE return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation			
TROKAY	TAOKAY	TCOKAY			
TRFAILED	TAENVIRO	TESYSERR TESTOP	TESUBSYS TETERM TERSOURC	TEDRAIN TEUNAUTH TENOTACT	
	TAFORMAT	TEBDOPCD TEBDTCB	TEBDXECB TEBDASCB	TEBDEPID	
	TAPROCED	TEAMODE	TEOWNER	TEINCMPL	TESTATE
	TATPLERR	TEACTIVE			
TRFATLFC	func. code	The function code loaded into register zero is invalid.			
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.			
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.			
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.			

Usage Information

The TCLOSE macro instruction is used to terminate (that is, close) an endpoint, and to remove any record of it from the system and communications domain in which it was created. Alternatively, the TCLOSE macro instruction can be used to pass control to an endpoint within the same task, or to another task or address space.

If OPTCD=DELETE is indicated, the TCLOSE macro instruction closes the endpoint, and removes any record of it from the system. All internal resources allocated to the endpoint are deleted, and the endpoint can no longer be referenced by any TPL-based macro instruction. Its existence in the communications domain, specified when the endpoint was created, is terminated.

An endpoint can be closed from any state. Normally, the application program closes an endpoint after any connection or association established at the endpoint has been released, and the local protocol address has been unbound. In this case, the state of the endpoint is opened (TSOPENED). However, if the endpoint is closed from any other state, the previously described operations are simulated to return the endpoint to the opened state before closing. If the endpoint was engaged in data transfer, any data buffered at the endpoint is discarded, and an abortive disconnect is executed. It is also possible to issue TCLOSE OPTCD=DELETE on the endpoint that is being passed by TCLOSE OPTCD=PASS.

The TCLOSE macro instruction provides a quick and convenient way to clean up after a major failure. In fact, the API resource recovery routines execute a TCLOSE function for each endpoint opened by a terminating task. The application program is responsible for recovering resources that it allocated such as TPL and data storage areas.

An endpoint can only be closed by the task that opened it. The opening task is said to control (or own) the endpoint, although other tasks can issue the API macro instructions that reference the endpoint. If it is necessary for another task to close an endpoint, control must be passed by indicating OPTCD=PASS when executing the TCLOSE macro instruction. The task acquiring control must issue a complementary TOPEN macro instruction indicating OPTCD=OLD, which specifies the same endpoint. The TCB and ASCB addresses of the new controlling task are returned to the application program.

Note: The endpoint identifier may change as control is passed from one task to another (see TOPEN). If control of an endpoint is passed to another task in the same address space, the task relinquishing control may continue to reference the endpoint, but must use the new endpoint identifier. The new endpoint ID is created and returned to the acquiring task after the complementary TOPEN macro instruction completes. The old endpoint ID should never be referenced again, either by the relinquishing or acquiring task. Please note that there are no restrictions of the order in which TCLOSE OPTCD=DELETE and TOPEN OPTCD=OLD occur.

Control can be passed to any task or address space that would otherwise have been authorized to create the endpoint.

- If a TCB address is indicated with the TCLOSE macro instruction, then only that task can acquire control
- If an ASCB address is indicated, then only a task in the designated address space can acquire control
- If neither of the above is indicated, then the first task in the current address space to issue the complementary TOPEN macro instruction acquires control of the endpoint

The endpoint can be passed in the opened (TSOPENED), disabled (TSDSABLD), enabled (TSENABLD), or connected (TSCONNCT) state, and must not have any incomplete requests pending at the endpoint. The state of the endpoint is unchanged, and the task relinquishing control may continue to reference the endpoint using the new endpoint ID created by the corresponding TOPEN OPTCD=OLD. Any exit list associated with the endpoint is replaced by the new exit list (if any) indicated by the complementary TOPEN macro instruction.

Whenever an asynchronous TCLOSE (OPTCD=ASYNC) completes normally (that is, recovery code is TAOKAY), the TCLOSE must be TCHECKED to let the cleanup of endpoint related control blocks complete. TOPEN specified with OPTCD=OLD does not complete until a TCHECK is performed on the related asynchronous TCLOSE OPTCD=PASS.

TCONFIRM

Acknowledge Confirm Indication — A connect confirm indication received at an endpoint as the result of a previous TCONNECT request being acknowledged with the TCONFIRM macro instruction. The remote protocol address of the connected (or associated) transport user is returned to the application program.

```
[ symbol ] TCONFIRM [ EP = endpoint_id ]
                    [ ,ADLEN = protocol_address_length ]
                    [ ,ADBUF = protocol_address_address ]
                    [ ,ADALET = protocol_address_alet
                    [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                                 [ ,SYNC | ASYNC ]
                                 [ ,TRUNC | NOTRUNC ]
                                 [ ,BLOCK | NOBLOCK ] ) ]
                    [, ECB = INTERNAL | event_control_block_addr ]
                    [ ,EXIT = tpl_exit_routine_address ]
                    [ ,MF = ( I | L | G | M | E ,[ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TCONFIRM macro instruction is to be executed.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

ADLEN = *protocol_address_length*

Length (in bytes) of the protocol address storage area identified by the ADBUF operand.

The length is updated when the request is completed to reflect the actual length of the protocol address returned. If the length is zero, the protocol address of the called transport user is not returned to the application program.

Default: Zero (return no protocol address).

ADBUF = *protocol_address_address*	Address of a storage area for returning the protocol address of the called transport user. The storage area should be large enough to contain the entire address. The format of the protocol address is provider-dependent, and its maximum size can be determined by issuing a TINFO macro instruction. The storage area can be aligned on any boundary. Default: Zero (no protocol address storage area).
ADALET = *protocol_address_alet*	Access List Entry Token (ALET) that is used in access register (AR) mode when referencing the storage specified by the ADBUF parameter. The ADALET value must be an ALET that is contained in the Dispatchable Unit Access List (DUAL) of the caller. The ADALET parameter may be used only if OPTCD=EXTEND is also specified. Default: Zero (the storage is contained in the address space of the caller).
OPTCD = SHORT | LONG | EXTEND	Format attribute of the parameter list associated with this request. OPTCD=SHORT a different control block identifier indicates that a subset of the TPL was generated. OPTCD=LONG a standard, full-length TPL is generated. OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters. Default: SHORT if MF=I or MF operand omitted, LONG otherwise.
OPTCD = SYNC | ASYNC	Synchronization mode to use when executing the TCONFIRM macro instruction. OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction completes. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API. OPTCD=ASYNC the request executes in asynchronous mode, and control is returned immediately after scheduling the TCONFIRM request. The application program is responsible for issuing the TCHECK macro instruction. Default: SYNC (synchronous mode).

OPTCD = TRUNC | NOTRUNC

Indicates whether the information returned to the application program by the transport provider should be truncated if it does not fit within the storage area provided.

OPTCD=TRUNC the excess is truncated, and the TCONFIRM macro instruction is completed conditionally as long as no other errors occur.

OPTCD=NOTRUNC nothing is placed in the storage area, and the TCONFIRM macro instruction is completed abnormally.

Default: NOTRUNC (no truncation).

OPTCD = BLOCK | NOBLOCK

Indicates whether the issuing task can be suspended if the TCONFIRM macro instruction cannot be completed immediately.

OPTCD=BLOCK (and no connect information has been received) the issuing task is suspended until the connection request is confirmed.

OPTCD=NOBLOCK the TCONFIRM macro instruction is completed immediately, and an abnormal return code indicates that the task would have been suspended for an indefinite period.

In either case, if a connect confirmation was received, the TCONFIRM macro instruction completes normally without suspending the issuing task.

When OPTCD=NOBLOCK is indicated, the TCONFIRM macro instruction can be used to poll for a connect confirmation. If the connection request was confirmed, the request is completed as usual. Otherwise, the request is completed abnormally, and the transport user can try again after delaying an appropriate period (for example, the expected round-trip time).

Default: BLOCK (suspend issuing task if necessary).

ECB = INTERNAL | *event_control_block_addr*

Location of an Event Control Block (ECB) to be posted by the API when the TCONFIRM macro instruction associated with this TPL is completed.

The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB.

The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB.

This operand is mutually exclusive with the following EXIT operand.

Default: INTERNAL (internal ECB).

EXIT = *tpl_exit_routine_address*

Address of an exit routine to be scheduled when the TCONFIRM macro instruction associated with this TPL is completed.

The TPL exit routine is scheduled only if asynchronous mode has been specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB.

This operand is mutually exclusive with the previous ECB operand.

Default: Not indicated (no TPL exit routine).

MF = (I | L | G | M | E , [*tpl_address*])

Standard, list, generate, modify, or execute form of the TCONFIRM macro instruction.

The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used.

See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: MF=I (standard, nonreentrant form).

Completion Information

The TCONFIRM macro instruction completes normally (or conditionally) when the confirm indication, generated by a connect confirmation sent by the called transport user, is received by the application program. The state of the endpoint is changed from connect-in-progress (TSOUCONN) to connected (TSCONNCT), and the endpoint is ready to send and receive data.

If a storage area was provided by the application program, the protocol address of the called transport user is returned. The protocol address length field indicated by macro instruction operand ADLEN is updated to reflect the actual length of the protocol address.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY), and a conditional completion code is returned in register zero. TCTRUNC is set if the protocol address returned to the application program was truncated to fit in the storage area provided. The TPL return code field is set accordingly. No other information is returned.

If the TCONFIRM macro instruction completes abnormally, no information is returned, and the pending connect confirmation (if any) remains queued. The state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TCONFIRM return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY	TCTRUNC	
TRFAILED	TAEXCPTN	TENOBLOK		
	TAINTEG	TEPROTO	TEOVRFLO	TEDISCON
	TAENVIRO	TESYSERR	TESUBSYS	TEDRAIN
		TESTOP	TETERM	
		TEUNSUPO		
		TEUNSUPF		
	TAFORMAT	TEBDFNCD	TEBDOPCD	TEBDECB
		TEBDEXIT	TEBDDATA	TEBDOPTN
	TAPROCED	TEAMODE	TESTATE	TEINCMPL
	TATPLERR	TEACTIVE		
TRFATLFC	func. code	The function code loaded into register zero is invalid.		
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.		
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.		
TRFATLAP	diag. code	The APCB associated wit the transport user is closed or is in the process of closing.		

Usage Information

The TCONFIRM macro instruction is used to receive a confirm indication that is pending for the endpoint. A confirm indication is generated when a connect confirmation is sent by a remote transport user in response to a connection request initiated by the application program with a TCONNECT macro instruction. On successful completion of this macro instruction, the specified endpoint is connected, and ready to send and receive data.

The protocol address of the remote transport user is returned to the application program.

Note: Existing protocols require that the called protocol address (the protocol address supplied with the TCONNECT request) and responding protocol address (the protocol address returned with TCONFIRM completion) be the same.

The TCONFIRM macro instruction is normally issued at an endpoint operating in connection mode. However, if the endpoint is operating in connectionless mode, and the application program created an association by issuing a TCONNECT macro instruction, the TCONFIRM macro instruction must be issued to receive the (simulated) connect confirmation generated by the API. Refer to *Assembler API Concepts* for a discussion of associations in connectionless mode.

TCONNECT

Request a Connection with another Transport User— The TCONNECT macro is used to request a connection to a remote transport user. The application program supplies the remote protocol address, protocol options, and any connect user data it wants to send to the remote transport user.

```
[ symbol ] TCONNECT [ EP = endpoint_id ]
                    [ ,ADLEN = protocol_address_length ]
                    [ ,ADBUF = protocol_address_address ]
                    [ ,ADALET = protocol_address_alet ]
                    [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                                 [ ,SYNC | ASYNC ] ) ]
                                 [ ,ECB = INTERNAL | event_control_block_addr ]
                    [ ,EXIT = tpl_exit_routine_address ]
                    [ ,MF = ( I | L | G | ME , [ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TCONNECT macro instruction is executed.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

ADLEN = *protocol_address_length*

Length (in bytes) of the protocol address contained in the storage area identified by the ADBUF operand. A length of zero is invalid, and causes the request to be abnormally completed.

Default: Zero (no protocol address).

ADBUF = *protocol_address_address*

Address of a storage area containing the protocol address of a remote transport user.

If the endpoint is operating in connection mode, a connection is established to the transport user at the indicated protocol address.

If the endpoint is operating in connectionless mode, an association is made such that any datagram sent with a TSEND macro instruction is sent to the indicated address, and a TRECV macro instruction acts as a filter selecting only those datagrams from the associated protocol address.

The length of the protocol address is designated by the ADLEN operand.

Default: Zero (no protocol address storage area).

ADALET = *protocol_address_alet*

Access List Entry Token (ALET) that is used in access register (AR) mode when referencing the storage specified by the ADBUF parameter.

The ADALET value must be an ALET that is contained in the Dispatchable Unit Access List (DUAL) of the caller. You can only use the ADALET parameter if OPTCD=EXTEND is also specified.

Default: Zero (the storage is contained in the address space of the caller).

OPTCD = SHORT | LONG | EXTEND

Format attribute of the parameter list associated with this request.

OPTCD=SHORT a different control block identifier is used to indicate that a subset of the TPL has been generated.

OPTCD=LONG a standard, full-length TPL is generated.

OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters.

Default: SHORT if MF=I or MF operand omitted, LONG otherwise.

OPTCD = SYNC | ASYNC

Synchronization mode to use when executing the TCONNECT macro instruction.

OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction is complete. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API.

OPTCD=ASYNC the request executes in asynchronous mode, and returns control immediately after scheduling the TCONNECT request. The application program is responsible for issuing the TCHECK macro instruction.

Default: SYNC (synchronous mode).

ECB = INTERNAL | *event_control_block_addr*

Location of an Event Control Block (ECB) to be posted by the API when the TCONNECT macro instruction associated with this TPL is completed.

The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB.

The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB.

This operand is mutually exclusive with the following EXIT operand.

Default: INTERNAL (internal ECB).

EXIT = *tpl_exit_routine_address*

Address of an exit routine to be scheduled when the TCONNECT macro instruction associated with this TPL is completed.

The TPL exit routine is scheduled only if asynchronous mode has been specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB.

This operand is mutually exclusive with the previous ECB operand.

Default: Not indicated (no TPL exit routine).

MF = (I | L | G | M | E , [*tpl_address*])

Standard, list, generate, modify, or execute form of the TCONNECT macro instruction.

The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used.

See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: MF=I (standard, nonreentrant form).

Completion Information

The TCONNECT macro instruction completes normally (or conditionally) when the connect-request primitive is issued to the transport provider. The state of the endpoint is changed from disabled (TSDSABLD) to connect-in-progress (TSOUCONN). The connection is not established until a confirm indication is received by the application program via a TCONFIRM macro instruction.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY), and a conditional completion code is returned in register zero. TCNEGOT is set in the conditional completion code if a requested protocol option was negotiated to an inferior value. The TPL return code field is set accordingly. No other information is returned.

If the TCONNECT macro instruction completes abnormally, no connection request is sent to the called transport user. The state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TCONNECT return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY	TCNEGOT	
TRFAILED	TAEXCPTN	TENONEGO		
	TAINTEG	TEPROTO		
	TAENVIRO	TESYSERR TETERM TERSOURC	TESUBSYS TESTOP TEUNSUPF	TEDRAIN TEUNSUPO TEUNAUTH
	TAFORMAT	TEBDFNCD TEBDEXIT TEBDOPTN	TEBDOPCD TEBDADDR	TEBDECB TEBDDATA
	TAPROCED	TEAMODE	TESTATE	TEINCMPL

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation
	TATPLERR	TEACTIVE
TRFATLFC	func. code	The function code loaded into register zero is invalid.
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.

Usage Information

The TCONNECT macro instruction is used to request a connection with a remote transport user. The protocol address of the called transport user, connect user data, and protocol options to associate with the connection are provided with the connection request. The endpoint must be in the disabled state (TSDSABLD) when the TCONNECT macro instruction is issued.

The TCONNECT macro instruction only serves to initiate the connection request. It is completed when the corresponding connect-request primitive is issued to the transport provider. The connection is not established until a connect confirmation is received from the called transport user. Receipt of a connect confirmation causes a confirm indication to be generated, and the application program can receive the indication with a TCONFIRM macro instruction.

The TCONNECT macro instruction is generally used by application programs operating in client mode. The protocol address (or name) of the remote server should be well known to the application program and other transport users that desire to use its services. Local directory services can be used to map host and service names into a protocol address that can then be supplied with the TCONNECT macro instruction. Directory services supported by TCPaccess are documented in the chapter “DNR Directory Services.”

The TCONNECT macro instruction is primarily intended for endpoints operating in connection mode. However, when executed at an endpoint operating in connectionless mode, a permanent association is made with the indicated transport user. Once this association is made, TSEND can be used in place of TSENDTO to transmit datagrams to the associated transport user. TRECV can be used in place of TRECVFR to select datagrams received from the same transport user.

An association is most useful when the application program is operating as a client using a connectionless-mode service. The TCONNECT macro instruction is used in the same way as with connection-mode service. The protocol address and options are saved and used for each subsequent send and receive operation. A TCONFIRM macro instruction is required to complete the association, just as in connection mode.

TDISCONN

Initiate Abortive Disconnect—The TDISCONN macro instruction is used to release a connection to a remote transport user, or to abandon a connection attempt that is in progress. The connection release is immediate, and no attempt is made to preserve data in transit.

```
[ symbol ] TDISCONN [ EP = endpoint_id ]
                    [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                                 [ ,SYNC | ASYNC ]
                                 [ ,ABORT | CLEAR ] ) ]
                    [ ,ECB = INTERNAL | event_control_block_addr ]
                    [ ,EXIT = tpl_exit_routine_address ]
                    [ ,MF = ( I | L | G | M | E , [ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TDISCONN macro instruction is to be executed.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

OPTCD = SHORT | LONG | EXTEND	Format attribute of the parameter list associated with this request. OPTCD=SHORT a different control block identifier used to indicate that a subset of the TPL was generated. OPTCD=LONG a standard, full-length TPL is generated. OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters. Default: SHORT if MF=I or if MF operand is omitted, LONG otherwise.
OPTCD = SYNC | ASYNC	Synchronization mode to use when executing the TDISCONN macro instruction. OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction is complete. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API. OPTCD=ASYNC the request executes in asynchronous mode, and returns control immediately after scheduling the TDISCONN request. The application program is responsible for issuing the TCHECK macro instruction. Default: SYNC (synchronous mode).
OPTCD = ABORT | CLEAR	Action that the transport provider should take when the application program issues a TDISCONN macro instruction at an endpoint for which a disconnect indication is already pending. OPTCD=ABORT the TDISCONN request clears the disconnect indication, and the macro instruction is completed normally (or conditionally) if no other errors occur. OPTCD=CLEAR the request is completed abnormally, and the application program must clear the pending disconnect indication with a TCLEAR macro instruction. Default: CLEAR (clear pending disconnect indication).

ECB = INTERNAL | *event_control_block_addr*	Location of an Event Control Block (ECB) to be posted by the API when the TDISCONN macro instruction associated with this TPL is completed. The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB. The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB. This operand is mutually exclusive with the following EXIT operand. Default: INTERNAL (internal ECB).
EXIT = *tpl_exit_routine_address*	Address of an exit routine to be scheduled when the TDISCONN macro instruction associated with this TPL has completed. The TPL exit routine is scheduled only if asynchronous mode is specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB. This operand is mutually exclusive with the previous ECB operand. Default: Not indicated (no TPL exit routine).
MF = (I | L | G | M | E , [*tpl_address*])	Standard, list, generate, modify, or execute form of the TDISCONN macro instruction. The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used. Refer to List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters. Default: MF=I (standard, nonreentrant form).

Completion Information

The TDISCONN macro instruction completes normally when the corresponding disconnect primitive is issued to the transport provider. Connection release is immediate. The state of the endpoint is changed to disabled (TSDSABLD) or enabled (TSENABLD) in accordance with the value of QLSTN specified in the TBIND macro instruction.

If the new state is:

- Enabled, the endpoint can resume receiving connect indication
- Disabled, the endpoint can be used to request another connection

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY). The conditional completion code in register zero is always zero (TCOKAY), and the TPL return code field is set accordingly. No other information is returned.

If the TDISCONN macro instruction completes abnormally, the connection is not released or abandoned. The state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TDISCONN return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY		
	TAINTEG	TEPROTO	TEDISCON	
	TAENVIRO	TESYSERR TESTOP TEUNSUPF	TESUBSYS TETERM	TEDRAIN TEUNSUPO
	TAFORMAT	TEBDFNCD TEBDEXIT	TEBDOPCD TEBDDATA	TEBDECB
	TAPROCED	TEAMODE	TESTATE	TEINCMPL

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation
	TATPLERR	TEACTIVE
TRFATLFC	func. code	The function code loaded into register zero is invalid.
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.

Usage Information

The TDISCONN macro instruction is used to request the immediate release of an established connection, or to abandon a pending connection request previously initiated with a TCONNECT macro instruction.

The TDISCONN macro instruction causes the abortive release of a connection. That is, the release is immediate, and any unsent data buffered at the endpoint is discarded. If it is important that all previously sent data reach the remote transport user, the application program should use an orderly release initiated with a TRELEASE macro instruction, or implement a session-layer protocol to gracefully terminate the session.

The TDISCONN macro instruction is normally issued when the endpoint is in the connected (TSCONNCT) state. The connection may have been established in client mode using the TCONNECT and TCONFIRM macro instructions, or established in server mode using the TLISTEN and TACCEPT macro instructions. When control is returned to the application program, the endpoint is considered disconnected, and is returned to the state that existed before the connection was established.

- If the application program is operating in client mode, the endpoint returns to the disabled (TSDSABLD) state, and another connection request can be initiated
- If the application program is operating in as a multithreaded server, the endpoint is also returned to the disabled state, and can be reused for accepting new connections

- If the application program is operating as a single-threaded server, the endpoint is returned to the enabled (TSENABLD) state, and listening for another connect indication can resume

The TDISCONN macro instruction can also be used to abandon a connection request that has not yet been confirmed. This situation may occur when the application program has issued a TCONNECT macro instruction, and has not received a connect confirmation.

Example

The remote transport user may be slow in responding, or the confirmation may be delayed because of congestion or other network problems. In this case, the endpoint must be in the connect-in-progress (TSOUCONN) state when the TDISCONN macro instruction is issued, and enters the disabled (TSDSABLD) state when the macro instruction completes.

Connected endpoints normally operate in connection mode. However, if an association was established with a remote transport user, the endpoint may be operating in connectionless mode, and the TDISCONN macro instruction may be issued to terminate this association.

The application program may then:

- Create a new association
- Begin sending and receiving datagrams with the TSENDTO and TRECVFR macro instructions
- Unbind the protocol address and close the endpoint

TDSECT

Generate the API Dummy Control Sections—The TDSECT macro instruction is used to generate DSECTs for the API data structures that must be created, managed, or referenced by the application program.

```
[ symbol ] TDSECT ( [,TEM]
                    [,TIB]
                    [,TPA]
                    [,TPL]
                    [,TPO]
                    [,TPL]
                    [,TPO]
                    [,TSW]
                    [,TUB]
                    [,TPO]
                    [,TXL]
                    [,TXP]
                    [,ALL] )
```

DSECT Types

A list of DSECTs (dummy control sections) that should be generated during assembly of the application program. Each operand is the name of an API data structure.

Valid operands:

TEM—Defines the structure and content of an Transport Error Message returned by the TERROR macro instruction. The information returned is formatted as a multi-line WTO parameter list, and can be supplied directly to a WTO macro instruction. The application program may use this DSECT to manipulate certain fields within the parameter list (for example, route codes and message descriptors).

TIB—Defines the structure and content of a Transport Protocol Information Block (TIB). The TIB contains basic transport protocol information returned by the TINFO macro instruction when OPTCD=PRIMARY is indicated. The format of this information is standard for all transport providers, and is intended to convey the basic characteristics of the underlying transport protocol and service.

TPA—Defines the structure and content of a Transport Protocol Address (TPA) in a particular communications domain. The domain can be specified with the DOMAIN operand, and should be consistent with domain specified when endpoints are opened (see TOPEN).

TPL—Defines the structure and content of a Transport Service Parameter List (TPL). The TPL is the primary API data structure for passing information between the application program and the API routines. All TPL-based macro instruction operands are stored or anchored in the TPL.

TPO—Defines the structure and content of Transport Protocol Options (TPO) supported by a particular transport provider. The transport provider is identified by the communications domain that it services, and is indicated by the DOMAIN operand on the TDSECT macro instruction. The value specified should be consistent with the communications domain specified when endpoints are opened (see TOPEN).

TSW—Defines the structure and content of a Transport Endpoint State Word (TSW). The TSW contains endpoint state information that is returned by the TSTATE macro instruction.

TUB—Defines the structure and content of a Transport Endpoint User Block (TUB). The TUB contains user ID, group name, and password information that are associated with an endpoint for the purpose of authorizing access to facilities, and accounting for their use The TUB may be provided as an argument of the TOPEN and TUSER macro instructions.

TXL—Defines the structure and content of a Transport Endpoint Exit List (TXL). The TXL may be provided as a parameter of an AOPEN or TOPEN macro instruction, and is used to identify exit routines that are to be entered for processing certain asynchronous events.

TXP—Defines the structure and content of a Transport Exit Parameter List (TXP). The TXP contains parameters and other information passed by the API to exit routines that are entered to process asynchronous events. The address of the TXP is loaded into register one when the exit routine is entered.

ALL—**Not** the name of a structure. It is an indication to generate all of the previous DSECTs as if each name had been indicated separately.

Note: These names should not appear in the operand list more than once, and should not be included on more than one TDSECT macro instruction.

Completion Information

The TDSECT macro instruction is declarative and does not generate any executable code. The indicated DSECTs are generated at the point in the application program where the TDSECT macro instruction occurs.

Return Codes

No return codes are generated.

Usage Information

The TDSECT macro instruction is used to generate dummy control sections (DSECTs) that map the API data structures. Each operand of the macro instruction is the:

- Name of an API data structure
- Label that should appear in a USING statement to establish addressability to the data structure

The API defines several data structures that are shared between the API routines and the application program. Some of these structures are created by the application program and referenced by the API, others are created by the API and referenced by the application program. In all cases, the API routines use the same DSECTs to access and manipulate these data structures. If the application program does likewise, information that is bound at assembly time can be easily changed by reassembly of the application program.

If the application program manipulates fields in the TPL directly instead of using the API macro instructions, the TPL DSECT should always be used to reference storage locations. In addition, whenever one of these data structures is created dynamically, the symbolic name for the length of the data structure should be used for allocating or reserving memory. The API always follows the convention that the symbolic name that specifies the length is the structure name appended with the characters LEN.

Example

The length of a standard format TPL is defined by the symbol TPLLEN.

The application program should be careful not to use any assembly language labels that conflict with labels defined by these DSECTs. The API labels always begin with the letter T.

TERROR

Analyze Error and Generate Error Message— The TERROR macro instruction is used to analyze an error associated with a previous TPL-based macro instruction, and to generate an error message describing the error that can be written to a log data set, or displayed to the system operator or local user.

```
[symbol ]  TERROR  [ VERBATIM | SUMMARY, ] MF = ( E, tpl_address )
```

VERBATIM | SUMMARY

Type of message to generate.

VERBATIM a literal message is generated that contains the actual values of TPL fields and other information that may be useful in diagnosing the error.

SUMMARY the TPL is analyzed and a message is generated that summarizes the error.

Default: SUMMARY (generate summary message).

MF = (E, *tpl_address*)

Execute form of the TERROR macro instruction.

The second sublist operand, *tpl_address*, specifies the address of the TPL associated with a request that completed abnormally.

Refer to List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: None (must be coded as indicated).

Completion Information

The TERROR macro instruction completes normally when an error message is formatted and ready to be output by the application program. The message is returned in a storage area allocated from subpool zero, and is formatted as a multi-line message compatible with the WTO and WTP macro instructions.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY). Register zero contains the address of a data structure (TEM) containing the error message. No information is stored in the TPL, and no other information is returned.

If the TERROR macro instruction completes abnormally, no error message is formatted or returned. The TPL or TPL address may be corrupted, and should not be reused. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field is not modified, and contains the information returned by the previous TPL-based macro instruction.

Note: The SYNAD or LERAD exit routines are not entered when the TERROR macro instruction completes abnormally.

Return Codes

The following table lists the symbolic names for the TERROR return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code
TROKAY	TEM address	n/a
TRFAILED	TATPLERR	n/a
	TAENVIRO	TEUNSUPO TERSOURC
TRFATLFC	func. code	n/a
TRFATLPL	diag. code	n/a
TRFATLAM	diag. code	n/a
TRFATLAP	diag. code	n/a

Usage Information

The TERROR macro instruction is used to generate an error message after a TPL-based macro instruction completes abnormally. The application program should display the message to the system or network operator, or log it for later inspection.

The following types of messages can be generated:

- Indicated by VERBATIM coded in the first positional operand field, contains diagnostic information extracted from the TPL and other API data structures. This message provides a verbatim description of the error.
- Indicated by SUMMARY, and consists of a summary message based on an analysis of the TPL. This is more appropriate for displaying to a non-technical user of the application program.

Note: The SUMMARY format is not implemented at this time.

The TERROR macro instruction should be executed after a TPL-based macro instruction completes abnormally with a general return code of four (TRFAILED), and a recovery action code less than 24 (TATPLERR). If the recovery action code was TATPLERR, no information was stored in the TPL return code field, and the error message generated is not meaningful. If the general return code was greater than four (TRFATLFC, TRFATLPL, TRFATLAM, and TRFATLAP), a fatal error occurred. Issuing the TERROR macro instruction in the latter case probably results in a similar error.

The message is returned in a storage area allocated from subpool zero. The message itself is formatted as a multi-line WTO parameter list. The TEM DSECT generated by the TDSECT macro instruction maps the storage area, and can be used to modify fields in the parameter list before a WTO or WTP macro instruction is issued. Alternatively, the application program can extract the message from the parameter list, and output it using whatever means is appropriate. The storage area should be freed when it is no longer required. The first fullword in the storage area contains the subpool and length of the storage area.

Example

This example shows how the TERROR macro instruction is intended to be used. It is assumed that register 1 contains the address of a TPL that completed abnormally:

```
TERROR VERBATIM,MF=(E,(1))
LTR 15,15
BNZ SKIPWTO
LR 2,0
XR 0,0

USING TEM,2
WTO MF=(E,TEMWTO)
L 0,TEMSL
FREEMAIN R,LV=(0),A=(2)
```

Format of a Verbatim Message

The format of a verbatim message is fixed. This is an example of verbatim message format:

```
T01API001I Transport endpoint error

        JOB jjjjjjjj STEP ssssssss APPL aaaaaaaa USER uuuuuuuu
        TPL xxxxxxxx APCB xxxxxxxx QLSN xxxxxxxx
        SEPM STAT xx TSTAT xx (xxxxxxxx)
        TPL IDENT xx FNCCD xx (ffffffff) ACTIV xx FLAGS xx
        TPL ACTCD xx ERRCD xx (eeeeeeee) DGNCD xxxx (mmmmmmmm)
        TPL EPID xxxxxxxx ECBXR xxxxxxxx OPTCD xxxxxxxx
        TPL PARM1 xxxxxxxx PARM2 xxxxxxxx PARM3 xxxxxxxx
        TPL ADBUF xxxxxxxx DABUF xxxxxxxx OPBUF xxxxxxxx
        TPL ADLEN xxxxxxxx DALEN xxxxxxxx OPLEN xxxxxxxx
```

Upper case fields are generated exactly as shown. Lower case fields are edited from information contained in the API data structures. Edit fields containing lower case *x*'s (for example, *xxxxxxxx*) represent hexadecimal values. All other fields contain alphanumeric character strings. Each line of the message is described separately:

Line 1 — Appears exactly as shown, and identifies the message as being a verbatim error message.

Line 2 — Contains the job name (JOB), step name (STEP), application name (APPL), and endpoint user name (USER) associated with the endpoint and TPL.

Line 3 — Contains the TPL address (TPL), APCB address (APCB), number of pending connect indications (QLSN).

Line 4 — Contains the endpoint state word consisting of the internal state (SEPM STAT) and the current TLI state (TSTAT).

ssssssss is the TSW symbolic name for the current state value.

Line 5 — Contains TPL fields consisting of the TPL control block identifier (IDENT), the function code (FNCCD), the active semaphore (ACTIV), and various flag bits (FLAGS).

ffffffff is the TPL symbolic name for the function code.

Line 6 — Contains the TPL return code consisting of the recovery action code (ACTCD), the specific error code (ERRCD), and the diagnostic code (DGNCD).

eeeeeeee is the TPL symbolic name for the specific error code

mmmmmmmm is the module name that generated the error (derived from diagnostic code).

Line 7	Contains TPL fields consisting of the endpoint identifier (EPID), the ECB or completion exit routine address (ECBXR), and option codes (OPTCD).
Line 8	Contains the three TPL fixed-length parameters (PARM1, PARM2, and PARM3).
Line 9	Contains the addresses of the three variable-length parameters–the protocol address (ADBUF), user data (DABUF), and protocol options (OPBUF).
Line 10	Contains the length of each variable-length parameter whose address appears in the line above (ADLEN, DALEN, and OPLEN).

The TERROR macro instruction should not be issued after a TOPEN failure. If this is attempted, unpredictable results occur.

TEVNTLST

Create an Event List— The TEVNTLST macro instruction associates an exit list with an endpoint via a TOPEN macro instruction or allows ECBs to be specified for protocol event and TPEND notifications.

```
TEVNTLST [,CONNECT = (address, ECB | EXIT)]
         [,CONFIRM = (address, ECB | EXIT)]
         [,DATA = (address, ECB | EXIT)]
         [,XDATA = (address, ECB | EXIT)]
         [,DGERR = (address, ECB | EXIT)]
         [,DISCONN = (address, ECB | EXIT)]
         [,RELEASE = (address, ECB | EXIT)]
         [,SENDWIND=(address, ECB | EXIT)]
         [,TPEND = (address, ECB | EXIT)]
         [,MF = ( L | M [, exit_list_address] ) ]
```

CONNECT = (*address,* ECB | EXIT)	First subparameter specifies the address of a routine to be entered or an ECB to be posted when certain asynchronous protocol events occur.
CONFIRM = (*address,* ECB | EXIT)	If an exit routine is used, the second parameter specifies whether the first subparameter is an ECB or an exit address. The address of a parameter list (mapped by the TXP DSECT) is passed to the routine in register one.
DATA = (*address,* ECB | EXIT)	
XDATA = (*address,* ECB | EXIT)	
DGERR = (*address,* ECB | EXIT)	An event code (TXPEVENT) stored in the parameter list by the API identifies the event when the same exit routine is used to handle more than one protocol event.
DISCONN = (*address,* ECB | EXIT)	
RE*LEASE = (address, ECB | EXIT)*	Refer to Event Codes for defined event codes.
SENDWIND = (address, ECB | EXIT)	Default: Zero (no exit routine).

TPEND = (*address,* ECB | EXIT)

First subparameter specified the address of a routine to be entered or an ECB to be posted when the transport provider terminates and can no longer provide service to the application program.

If an exit routine is used, the second parameter specifies whether the first subparameter is an ECB or an exit address. The address of a parameter list (mapped by the TXP DSECT) is passed to the routine in register one.

A reason code (TXPREASN) stored in the parameter list by the API identifies the reason for termination of service.

Refer to Event Codes for defined event codes.

Default: Zero (no TPEND exit routine).

MF = (L | M [*exit_list_address*])

List or modify form of the TEVNTLST macro instruction. The second sublist operand, *exit_list_address*, is the address of a storage area that contains (MF=M), or will contain (MF=L), the exit list (TXL).

If the exit list address is not provided, or the MF operand is not coded, the exit list is generated in line with the macro instruction.

See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: Not indicated (nonreentrant, inline list).

Event Codes

This section lists the event codes passed to the CONNECT, CONFIRM, DATA, XDATA, DGERR, DISCONN, RELEASE, and SENDWIND exit routines.

Name	Dec	Hex	Exit	Protocol Event
TXPECONN	0	X'	CONNECT	Connect indication.
TXPECONF	4	X'	CONFIRM	Confirm indication.
TXPEDATA	8	X'	DATA	Normal data indication.
TXPEXPDT	12	XC'	XDATA	Expedited data indication.
TXPERROR	16	X'	DGERR	Datagram error indication.
TXPEDISC	20	X'	DISCONN	Disconnect indication.
TXPERLSE	24	X'	RELEASE	Orderly release indication.
TXPESWIN	28	X'12'	SENDWIND	Send Window Opened.

TPEND Reason Codes

The following table lists the reason codes for TPEND calls. A reason code (TXPREASN) stored in the parameter list by the API identifies the reason for service termination.

Name	Dec	Hex	Explanation
TXPRDRAN	0	X'	Operator drained subsystem .
TXPRSTOP	4	X'	Operator stopped subsystem.
TXPRTERM	8	X'	Subsystem abnormally terminated.

Completion Information

If the MF operand is not coded, or MF=L is indicated (without TXL address), the exit list is generated at assembly time, and no executable code is expanded.

Otherwise, the macro instruction expansion contains executable code to generate or modify the exit list at the location specified by the application program. The general return code returned in register 15 is always zero (TROKAY). No other information is returned.

Return Codes

No error codes are generated.

Usage Information

The TEVNTLST macro instruction builds a list of the address supplied by the TEVNTLST operands. Each address identifies an application program routine that will be given control, or ECB to post, when the respective event occurs.

The list created by TEVNTLST is referenced by the TOPEN parameter EVENTLST. The structure of the event list is the same as the exit list built by the TOPEN form of TEXLST. An additional set of flags indicates whether each address is the address of an exit routine or an ECB.

Empty slots are created for operands that are not coded on the TEVNTLST macro instruction. Empty slots are set to zero, indicating that no exit routine was specified. The application program can use the modify form of the TEVNTLST macro instruction to update an event list after it is created. The event list must be aligned on a fullword boundary.

The address of the event list is provided as a parameter to the TOPEN macro instruction. When the endpoint is opened, the event list is permanently linked to the transport user or endpoint. When an event occurs that may require processing by the application program, the event list linked to the endpoint is checked first to see if an exit routine or ECB is defined. If so, the exit routine is entered to process the event or the respective ECB is posted. Otherwise, the exit list linked to the APCB is checked. If no exit routine or ECB is defined, the occurrence of the event must be detected by some other means (generally via return codes stored in the TPL).

When the APCB or endpoint is opened, a copy of the event list or exit list is saved. Therefore, exit routines and ECBs associated with a transport user or any one of its endpoints cannot be changed once the APCB is opened, or the endpoint created. However, if control of an endpoint is passed to another task or address space, a new event list can be specified via the TOPEN macro instruction that acquires control of the endpoint.

See TCPaccess *Assembler API Concepts* for a detailed discussion of exit routines and a listing of TXL DSECT that maps the event list.

TEXEC

Execute a Transport Service Parameter List— A Transport Service Parameter List (TPL) that was initialized, or used to make a previous request, can be executed or reexecuted using the TEXEC macro instruction. Issuing the TEXEC macro instruction with a function code is functionally equivalent to issuing the macro instruction indicated by the function code.

The TEXEC macro instruction accepts all operands defined for other TPL-based macro instructions, plus those defined in this topic. The precise definition and use of a particular operand depends on the value indicated for FNCCD. Refer to the macro instruction description for the indicated function to determine which operands can be coded, and how they are used.

```
[ symbol ] TEXEC [ EP = endpoint_id ]
                 [ ,ADLEN = protocol_address_length ]
                 [ ,ADBUF = protocol_address_address ]
                 [ ,ADALET = protocol_address_alet ]
                 [ ,DALEN = user_data_length ]
                 [ ,DABUF = user_data_address ]
                 [ ,DAALET = user_address_alet ]
                 [ ,OPLEN = protocol_options_length ]
                 [ ,OPBUF = protocol_options_address ]
                 [ ,OPALET = protocol_options_alet ]
                 [ ,QLSTN = listen_queue_length ]
                 [ ,NEWEP = new_endpoint_id ]
                 [ ,SEQNO = sequence_number ]
                 [ ,USER = endpoint_userid ]
                 [ ,TCB = task_control_block_address ]
                 [ ,ASCB = address_space_control_block_address ]
                 [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                              [ ,SYNC | ASYNC ]
                              [ ,TRUNC | NOTRUNC ]
                              [ ,NEGOT | NONEGOT ]
                              [ ,BLOCK | NOBLOCK ]
                              [ ,ASSIGN | USE ]
                              [ ,LOCAL | REMOTE ]
                              [ ,PRIMARY | SECNDRY | STATS ]
                              [ ,DECLARE | VERIFY | QUERY | DEFAULT ]
                              [ ,TP | API ]
                              [ ,MORE | NOMORE ]
                              [ ,NORMAL | EXPEDITE ]
                              [ ,EOM | NOTEOM ]
                              [ ,DIRECT | INDIR ]
                              [ ,ABORT | CLEAR ]
                              [ ,DELETE | PASS ]
                              [ ,TUB | ACEE ]
                              [ ,PLAIN | CIPHER ]
                              [ ,MBUF | NOMBUF ]
                              [ ,FULL | NOFULL ]
                              [ ,TIMEOUT | NOTIMEOUT ] ) ]
                 [ ,FNCCD = TACCEPT | TADDR | TBIND | TCLEAR |
                            TCLOSE | TCONFIRM | TCONNECT |
                            TDISCONN | TINFO | TLISTEN | TOPTION |
                            TRECV | TRECVERR | TRECVFR | TREJECT |
                            TRELACK | TRELEASE | TRETRACT | TSEND |
                            TSENDTO | TUNBIND | TUSER
                 [ ,ECB = INTERNAL | event_control_block_addr ]
                 [ ,EXIT = tpl_exit_routine_address ]
                 [ ,MF = ( G | E , tpl_address ) ]
                 [ ,MF = ( I | G | E , [ tpl_address ] ) ]
```

FNCCD = *function_code*

The API function to execute.

Valid values are:

TACCEPT	Accept connection request.
TADDR	Get protocol address.
TBIND	Bind local protocol address.
TCLEAR	Clear disconnect indication.
TCLOSE	Close endpoint.
TCONFIRM	Receive connect confirmation.
TCONNECT	Initiate connection request.
TDISCONN	Initiate abortive disconnect.
TINFO	Get transport protocol information.
TLISTEN	Listen for connect indications.
TOPTION	Endpoint option management.
TRECV	Receive from connected transport user.
TRECVERR	Receive datagram error indication.
TRECVFR	Receive a datagram.
TREJECT	Reject connection request.
TRELACK	Acknowledge orderly release indication.
TRELEASE	Initiate or complete orderly release.
TRETRACT	Retract a pending TLISTEN request.
TSEND	Send to connected transport user.
TSENDTO	Send a datagram.
TUNBIND	Unbind local protocol address.
TUSER	Associate user with endpoint.

If a function code is specified, the definition and use of other operands is determined by the designated function.

The operands that may be coded, and the rules that apply, are the same as those defined for the API macro instruction that corresponds to the function code. If a function code is not specified, the value stored in the TPL designates the function to execute.

Default: Not indicated (use function code in TPL).

MF = (I | G | E , [*tpl_address*])

Standard, generate, or execute form of the TEXEC macro instruction.

The second sublist operand, *tpl_address*, is the address of a storage area that contains (MF=E), or contains (MF=G), the Transport Function Parameter List (TPL). If the MF operand is not coded, an inline list and subroutine linkage is generated.

If the list or modify form is desired, use the TPL macro instruction.

See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: MF=I (standard, nonreentrant form)

Completion Information

Completion information is determined by the API function executed, and is defined in the section describing the macro instruction corresponding to the function executed.

In general, on normal return to the application program, register 15 contains zero (TROKAY) and register zero contains a conditional completion code, or zero (TCOKAY) if there was no conditional completion. On abnormal return, register 15 contains the general return code (unless modified by the SYNAD or LERAD exit routine), and register zero contains the recovery action code. The recovery action code and a specific error code may also be stored in the return code field of the TPL.

Return Codes

The following table lists the symbolic names for the TEXEC return codes. The values associated with the symbolic names can be found in the TPL macro expansion. These return codes are common to all API TPL-based macro instructions:

Note: For a description of the return codes that apply to a specific function, refer to the description of the macro instruction that corresponds to the function.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY		
TRFAILED	TAENVIRO	TESYSERR TESTOP	TESUBSYS TETERM	TEDRAIN
	TAFORMAT	TEBDFNCD TEBDEXIT	TEBDOPCD	TEBDECB
	TATPLERR	TEACTIVE		
TRFATLFC	func. code	The function code loaded into register zero is invalid.		
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.		
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.		
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.		

Usage Information

The TEXEC macro instruction is used to execute a Transport Service Parameter List (TPL) that was generated, or to reexecute a TPL that was been previously executed. The TEXEC macro instruction is complementary with the TPL macro instruction used to generate or modify a TPL.

The API macro instructions use the same inner macro instructions to generate the macro expansion. Therefore, except for functions performed in the outer most macro instruction, the TEXEC macro instruction used with a function code is functionally equivalent to the macro instruction that corresponds to the particular function.

Example 1

The macro instruction TEXEC QLSTN=5,FNCCD=TBIND,MF=(E,BINDTPL) generates the same expansion as TBIND QLSTN=5,MF=(E,BINDTPL).

In fact, the TEXEC macro instruction can be used to generate any TPL-based request other than TERROR, TCHECK, TSTATE, and TOPEN.

Most of the function-dependent validity checking is performed by the outer macro instruction (TBIND in the previous example). Therefore, the application programmer should be cautious when using the TEXEC macro instruction to make arbitrary requests.

Example 2

This macro instruction expands successfully at assembly time, but probably completes abnormally at runtime:

```
TEXEC QLSTN=5,FNCCD=TRECV,MF=(E,BINDTPL)
```

Whereas, this equivalent macro instruction fails at assembly time:

```
TRECV QLSTN=5,MF=(E,BINDTPL)
```

TEXLST

Create an Exit List— The TEXLST macro instruction is used to create an exit list that can be associated with a transport user via an AOPEN macro instruction, or associated with an endpoint via a TOPEN macro instruction.

```
[ symbol ] TEXLST [ AOPEN | TOPEN ]
                  [ ,SYNAD = exit_routine_address ]
                  [ ,LERAD = exit_routine_address ]
                  [ ,CONNECT = exit_routine_address ]
                  [ ,CONFIRM = exit_routine_address ]
                  [ ,DATA = exit_routine_address ]
                  [ ,XDATA = exit_routine_address ]
                  [ ,DGERR = exit_routine_address ]
                  [ ,DISCONN = exit_routine_address ]
                  [ ,RELEASE = exit_routine_address ]
                  [ ,SENDWIND = exit_routine_address ]
                  [ ,TPEND = exit_routine_address ]
                  [ ,APEND = exit_routine_address ]
                  [ ,MF = ( L | M [ ,exit_list_address ] ) ]
```

AOPEN | TOPEN

Indicates whether the exit list is associated with a transport user or a particular endpoint.

AOPEN the exit list is linked to the APCB with an AOPEN macro instruction.

TOPEN the exit list is linked to an endpoint via a TOPEN macro instruction. Since the exit routines associated with an endpoint are a subset of those defined for the APCB, this operand also determines the maximum length of the exit list.

Default: AOPEN (exit list linked via APCB).

SYNAD = *exit_routine_address*

Address of a routine to be entered if a physical error or other unusual condition occurs during the processing of a TPL-based request. Invalid requests and logic errors are handled by the LERAD exit routine.

A recovery action code is passed to the exit routine in register zero, and a copy is stored in the return code field of the TPL associated with the request. A specific error code is also stored in the return code field. The address of the TPL is passed in register one.

See Recovery Action Codes for SYNAD Routine Entry for the recovery action codes that cause the SYNAD routine to be entered.

Default: Zero (no SYNAD exit routine).

LERAD = *exit_routine_address*

Address of a routine entered when the application program issues a TPL-based request that results in a logic error. Physical errors or other unusual conditions are handled by the SYNAD exit routine.

A recovery action code is passed to the exit routine in register zero, and a copy is stored in the return code field of the TPL associated with the request. A specific error code is also stored in the return code field.

However, if the recovery action code is 24 (TATPLERR), the TPL may be active or corrupted, and no information is stored in the return code field. The address of the TPL is passed in register one. See Recovery Action Codes For LERAD Routine Entry for a list of recovery action codes that cause the LERAD routine to be entered.

Default: Zero (no LERAD exit routine).

CONNECT = *exit_routine_address*
CONFIRM = *exit_routine_address*
DATA = *exit_routine_address*
XDATA = *exit_routine_address*
DGERR = *exit_routine_address*
DISCONN = *exit_routine_address*
RELEASE = *exit_routine_address*
SENDWIND = *exit_routine_address*

Address of a routine to enter when certain asynchronous protocol events occur.

The address of a parameter list (mapped by the TXP DSECT) is passed to the routine in register one. An event code (TXPEVENT) stored in the parameter list by the API identifies the event when the same exit routine is used to handle more than one protocol event.

Refer to Event Codes for defined event codes.

Default: Zero (no exit routine) .

TPEND = *exit_routine_address*

Address of a routine to enter when the transport provider terminates and can no longer provide service to the application program.

The address of a parameter list (mapped by the TXP DSECT) is passed to the routine in register one. A reason code (TXPREASN) stored in the parameter list by the API identifies the reason for termination of service.

See TPEND Reason Codes to view a list of defined reason codes.

Default: Zero (no TPEND exit routine).

APEND = *exit_routine_address*

Address of a routine to enter when the API subsystem terminates and can no longer provide service to the application program.

The address of a parameter list (mapped by the TXP DSECT) is passed to the routine in register one. A reason code (TXPREASN) stored in the parameter list by the API identifies the reason for termination of service.

Default: Zero (no APEND exit routine).

MF = (L | M , [*exit_list_address*])

List or modify form of the TEXLST macro instruction.

The second sublist operand, *exit_list_address*, is the address of a storage area that contains (MF=M), or will contain (MF=L), the exit list.

If the exit list address is not provided, or the MF operand is not coded, the exit list is generated in line with the macro instruction.

Refer to List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: Not indicated (nonreentrant, inline list).

Recovery Action Codes

The following tables list the recovery action codes for SYNAD and LERAD routine entry. The values associated with the symbolic names can be found in the TPL macro expansion. These return codes are common to all API TPL-based macro instructions:

Recovery Action Codes for SYNAD Routine Entry

Name	Dec	Hex	Explanation
TAEXCTPN	4	X'	Exceptional condition.
TAINTEG	8	X'	Connection or data integrity error.
TAENVIRO	12	XC'	Environmental condition.

Recovery Action Codes For LERAD Routine Entry

Name	Dec	Hex	Explanation
TAFORMAT	16	X'	Format or specification error.
TAPROCED	20	X'	Sequence or procedural error.
TATPLERR	24	X'	Logic error with no TPL return code.

Event Codes

The following table lists the event codes passed to the CONNECT, CONFIRM, DATA, XDATA, DGERR, DISCONN, RELEASE, and SENDWIND exit routines. The values associated with the symbolic names can be found in the TXP macro expansion.

Name	Dec	Hex	Exit	Protocol Event
TXPECONN	0	X'	CONNECT	Connect indication.
TXPECONF	4	X'	CONFIRM	Confirm indication.
TXPEDATA	8	X'	DATA	Normal data indication.
TXPEXPDT	12	XC'	XDATA	Expedited data indication.
TXPERROR	16	X'	DGERR	Datagram error indication.
TXPEDISC	20	X'	DISCONN	Disconnect indication.
TXPERLSE	24	X'	RELEASE	Orderly release indication.
TXPESWND	28	X’18’	SENDWIND	Send Window opened.

APEND and TPEND Reason Codes

The following tables list the reason codes for TPEND and APEND calls. A reason code (TXPREASN) stored in the parameter list by the API identifies the reason for termination of service.

TPEND Reason Codes

Name	Dec	Hex	Explanation
TXPRDRAN	0	X'	Operator drained subsystem.
TXPRSTOP	4	X'	Operator stopped subsystem.
TXPRTERM	8	X'	Subsystem abnormally terminated.

APEND Reason Codes

Name	Dec	Hex	Explanation
TXPRDRAN	0	X'	Operator drained subsystem.
TXPRSTOP	4	X'	Operator stopped subsystem.
TXPRTERM	8	X'	Subsystem abnormally terminated.

Completion Information

If the MF operand is not coded, or MF=L is indicated (without *exit_list_address* (TXL) address), the exit list is generated at assembly time, and no executable code is expanded.

Otherwise, the macro instruction expansion contains executable code to generate or modify the exit list at the location specified by the application program. The general return code returned in register 15 is always zero (TROKAY). No other information is returned.

Return Codes

No error codes are generated.

Usage Information

The TEXLST macro instruction builds a list of exit routine addresses. Each operand in this macro instruction represents a class of events for which an exit routine can be entered by the API.

The address supplied for each operand identifies an application program routine to be given control when a particular event occurs.

Example

The SYNAD operand supplies the address of a routine that handles exceptional or unusual conditions (other than logic errors) for TPL-based macro instructions, and the CONNECT operand supplies the address of a routine that receives connect indications.

The length of the exit list depends on whether it is used with the AOPEN or TOPEN macro instruction. Some exit routines defined in the AOPEN exit list cannot be specified in a TOPEN exit list. Therefore, an AOPEN exit list is longer than a TOPEN exit list. A length parameter generated at the start of an exit list indicates the type of exit list, and is validity checked at execution time when referenced with an AOPEN or TOPEN macro instruction.

Empty slots are created for operands that are not coded on the TEXLST macro instruction. Empty slots are set to zero, indicating that no exit routine was specified. The application program can use the modify form of the TEXLST macro instruction to update an exit list after it is created. The exit list must be aligned on a fullword boundary.

The address of the exit list is provided as a parameter to the AOPEN or TOPEN macro instruction. When the APCB or endpoint is opened, the exit list is permanently linked to the transport user or endpoint. When an event occurs that may require processing by the application program, the exit list linked to the endpoint is checked first to see if an exit routine is defined. If so, the exit routine is entered to process the event. Otherwise, the exit list linked to the APCB is checked. If no exit routine is defined, the occurrence of the event must be detected by some other means (generally via return codes stored in the TPL).

When the APCB or endpoint is opened, a copy of the exit list is saved. Therefore, exit routines associated with a transport user or any one of its endpoints cannot be changed once the APCB is opened, or the endpoint created. However, if control of an endpoint is passed to another task or address space, a new exit list can be specified via the TOPEN macro instruction that acquires control of the endpoint.

Refer to the TCPaccess *Planning Guide* for a detailed discussion of exit routines. The exit list is mapped by the TXL DSECT, which is also listed in the appendix "Data Structures (Assembler Language)" in the TCPaccess *Assembler API Macros* guide.

TINFO

Retrieve Transport Protocol Information—Protocol information associated with an endpoint and maintained by the transport provider can be retrieved using the TINFO macro instruction.

```
[ symbol ] TINFO [ EP = endpoint_id ]
                 [ ,DALEN = protocol_information_length ]
                 [ ,DABUF = protocol_information_address ]
                 [ ,DAALET = protocol_information_alet ]
                 [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                              [ ,SYNC | ASYNC ]
                              [ ,TRUNC | NOTRUNC ]
                              [ ,PRIMARY | SECNDRY | STATS ] ) ]
                 [ ,ECB = INTERNAL | event_control_block_addr ]
                 [ ,EXIT = tpl_exit_routine_address ]
                 [ ,MF = ( I | L | G | M | E , [ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TINFO macro instruction is to be executed.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

DALEN = *protocol_information_length*

Length (in bytes) of the protocol information storage area identified by the DABUF operand. The length is updated when the request is completed to reflect the actual length of protocol information returned.

If the length is zero, the API macro instruction completes abnormally.

Default: Zero (return no protocol information).

DABUF = *protocol_information_address*

Address of a storage area for returning protocol information maintained for the designated endpoint. The storage area should be large enough to contain the requested information, and can be aligned on any boundary convenient to the application program.

The type of information requested is indicated by the OPTCD operand. Only the information indicated by OPTCD=PRIMARY is standardized for all transport providers. All other information types are provider-dependent. The information provided with a OPTCD=PRIMARY request can be used to determine the maximum size of the information unit returned when designating other information types.

Default: (no protocol information storage area).

DAALET = *protocol_information_alet*	Access List Entry Token (ALET) used in access register (AR) mode when referencing the storage specified by the DABUF parameter. The DAALET value must be an ALET that is contained in the Dispatchable Unit Access List (DUAL) of the caller. The DAALET parameter can be used only if OPTCD=EXTEND is also specified. Default: Zero (the storage is contained in the address space of the called).
OPTCD = SHORT | LONG | EXTEND	Format attribute of the parameter list associated with this request. OPTCD=SHORT a different control block identifier is used to indicate that a subset of the TPL has been generated. OPTCD=LONG a standard, full-length TPL is generated. OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters. Default: SHORT if MF=I or MF operand omitted, LONG otherwise.
OPTCD = SYNC | ASYNC	Synchronization mode to use when executing the TINFO macro instruction. OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction is complete. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API. OPTCD=ASYNC the request executes in asynchronous mode, and returns control immediately after scheduling the TINFO request. The application program is responsible for issuing the TCHECK macro instruction. Default: SYNC (synchronous mode).

OPTCD = TRUNC | NOTRUNC

Indicates whether the protocol information returned to the application program by the transport provider should be truncated if it does not fit within the storage area provided.

OPTCD=TRUNC the excess is truncated, and the TINFO macro instruction is completed conditionally as long as no other errors occur.

OPTCD=NOTRUNC nothing is placed in the storage area, and the TINFO macro instruction completes abnormally.

Default: NOTRUNC (no truncation).

OPTCD = PRIMARY | SECNDRY | STATS

Type of information requested.

PRIMARY—Designates primary protocol information whose format and meaning is standardized for all transport providers. The application program can use this information to determine the basic characteristics of the transport service and limits of the transport provider.

SECNDRY—Designates secondary protocol information whose format and meaning is specific to the transport service being used. This information includes internal protocol and state variables that govern the operation of the transport protocol. Transport providers are not required to support this option code. Transport providers in the current implementation do not support SECNDRY.

STATS—Designates statistical information recorded by the transport provider whose format and meaning is specific to the transport service being used. Transport providers are not required to support this option code. Transport providers in the current implementation do not support STATS.

Default: PRIMARY (return basic protocol information).

ECB = INTERNAL | *event_control_block_addr*	Location of an Event Control Block (ECB) to be posted by the API when the TINFO macro instruction associated with this TPL is completed. The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB. The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB. This operand is mutually exclusive with the following EXIT operand. Default: INTERNAL (internal ECB).
EXIT = *tpl_exit_routine_address*	Address of an exit routine scheduled when the TINFO macro instruction associated with this TPL is completed. The TPL exit routine is scheduled only if asynchronous mode is specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB. This operand is mutually exclusive with the previous ECB operand. Default: Not indicated (no TPL exit routine).
MF = (I | L | G | M | E , [*tpl_address*])	Standard, list, generate, modify, or execute form of the TINFO macro instruction. The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used. Refer to List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters. Default: MF=I (standard, nonreentrant form),

Completion Information

The TINFO macro instruction completes normally (or conditionally) when the requested protocol information is returned in the storage area provided by the application program. The length of the storage area is updated to reflect the actual amount of information returned. The state of the endpoint is unchanged.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY), and a conditional completion code is returned in register zero. TCTRUNC is set if the protocol information returned to the application program was truncated to fit in the storage area provided. The TPL return code field is set accordingly. No other information is returned.

If the TINFO macro instruction completes abnormally, no protocol information is returned to the application program. The state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field can also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TINFO return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY	TCTRUNC	
TRFAILED	TAINTEG	TEOVRFLO		
	TAENVIRO	TESYSERR TESTOP	TESUBSYS TETERM	TEDRAIN TEUNSUPO
	TAFORMAT	TEBDFNCD TEBDEXIT	EBDOPCD TEBDDATA	TEBDECB
	TAPROCED	TEAMODE	TEINCMPL	
	TATPLERR	TEACTIVE		
TRFATLFC	func. code	The function code loaded into register zero is invalid.		

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.

Usage Information

The TINFO macro instruction is used to get information from the transport provider that is maintained for a specific endpoint. An option code is used to indicate what type of information is desired. The information types range from basic protocol information that is common to all transport providers, to detailed protocol information and statistics that are provider-dependent.

The basic protocol information is standardized for all transport providers, and is requested by indicating PRIMARY with the OPTCD operand. The information returned can be used by the application program to determine basic characteristics and limits of the transport service.

Maximum lengths of protocol addresses, connect and disconnect user data, and protocol options are provided.

If storage areas used for returning such information by other macro instructions are allocated dynamically using this information, an overflow condition should not occur.

Basic Protocol Information Returned

The basic protocol information returned is fixed in length (76 bytes), and is mapped by the Transport Information Block (TIB) DSECT. The following table defines the information returned:

Note: Each field is designated by the label defined in the TIB DSECT that should be used to access it.

TIBTSDOM

A one-byte value defining the communications domain within which the endpoint was created.

The value corresponds to the communications domain requested by the TOPEN macro instruction. If none was explicitly requested, this value indicates the domain that was assigned based on system and installation default values.

TIBTSTYP

A one-byte value defining the mode of service assigned for the endpoint.

The value corresponds to the mode of service requested by the TOPEN service function. If none was explicitly requested, this value indicates the mode of service that was assigned based on system and installation default values.

TIBTSCHR

A flag byte defining one of these various characteristics of the transport service:

- Message boundaries are preserved within data stream
- Expedited data is supported
- The transport provider supports the protocol options parameter
- User data can be sent with connection request
- User data can be sent with disconnect request

TIBTSOPT

A flag byte indicating one of these additional options supported by the transport provider:

- Datagram address associations
- Orderly release
- Secondary protocol information
- Statistical information

TIBSYSID

Four-character ID of the MVS subsystem containing the transport service provider (or its surrogate).

The value is the same as the subsystem ID requested by the AOPEN service function. If none was explicitly requested, this value is the ID of the MVS subsystem that was assigned based on system and installation default values.

TIBSVCID — Eight-character ID of the transport service provider managing the endpoint.

The value is the same as the transport service ID requested by the TOPEN service function. If none was provided, this value is the ID of the transport service provider that was assigned based on system and installation default values.

TIBPROTO — Protocol number of the protocol used to provide the transport service.

The value is the same as the protocol number requested by the TOPEN service function. If none was provided, this value is the protocol number of the protocol that was assigned based on system and installation default values.

Transport Service Limits

The API uses a general notation for defining limits of the transport service. The limit of a particular facility is represented by a signed, integer value.

- If the value is greater than zero, the facility is supported by the transport provider, and the value is the limit of the facility
- If the value is -1, the facility is supported, but there is no limit
- If the value is -2, the facility is not supported at all

A value of zero is sometimes used to represent special characteristics of the facility.

Transport Interface Limits

TIBQLSTN — A value greater than zero indicates the maximum number of connect indications that can be queued by the transport interface.

-1 Indicates that there is no limit on the size of the connect indication queue.

-2 Specifies that the transport interface does not support the queueing of connect indications.

0 Not returned.

TIBQSEND — A value greater than zero indicates the maximum number of uncompleted send requests that can be queued by the transport interface.

-1 Indicates that there is no limit on the number of send requests. Since the transport interface must allow at least one uncompleted send request.

-2 or 0 Never returned.

TIBQRECV

A value greater than zero indicates the maximum number of uncompleted receive requests that can be queued by the transport interface.

-1 Indicates that there is no limit on the number of receive requests. Since the transport interface must allow at least one uncompleted receive request.

-2 or 0 Never returned.

TIBLTSND

A value greater than zero indicates the maximum number of user data bytes that can be transferred by the transport interface with a single send request.

-1 Indicates that there is no limit on the amount of data in a single send request. The transport interface always supports the sending of data.

-2 or 0 Never returned.

TIBLTRCV

A value greater than zero indicates the maximum number of user data bytes that can be transferred by the transport interface with a single receive request.

-1 Indicates that there is no limit on the amount of data in a single receive request. The transport interface always supports the receiving of data.

-2 or 0 Never returned.

TIBLSEND

A value greater than zero indicates the maximum number of user data bytes that can be pending for uncompleted send requests.

-1 Indicates that there is no limit on the total amount of pending send data.

-2 or zero Never returned.

TIBLRECV

A value greater than zero indicates the maximum number of user data bytes that can be pending for uncompleted receive requests.

-1 Indicates that there is no limit on the total amount of pending receive data.

-2 or 0 Never returned.

Transport Provider Limits

TIBLADDR — A value greater than zero indicates the maximum size of a transport protocol address.

- -1 Indicates that there is no limit on the address size.
- -2 Indicates that the transport provider does not provide user access to transport protocol addresses.
- 0 Is not returned by the transport provider.

TIBLOPTN — A value greater than zero indicates the maximum number of bytes in the protocol options parameter supported by the transport provider.

- -1 Indicates that there is no limit on the size of protocol options.
- -2 Specifies that the transport provider does not support user-specified protocol options.
- 0 Is not returned by the transport provider.

TIBLTSDU — A value greater than zero indicates the maximum size of a Transport Service Data Unit (TSDU).

- -1 Indicates that there is no limit on the size of a TSDU.
- -2 Indicates that the transfer of normal data is not supported by the transport provider.
- 0 Indicates that the transport provider does not support the concept of a TSDU, although it does support the sending of a data stream with no logical boundaries preserved across the connection.

TIBLXPDT — A value greater than zero indicates the maximum size of an Expedited Transport Service Data Unit (ETSDU).

- -1 Indicates that there is no limit on the size of an ETSDU.
- -2 Indicates that the transfer of expedited data is not supported by the transport provider.
- 0 Indicates that the transport provider does not support the concept of an ETSDU, although it does support the sending of an expedited data stream with no logical boundaries preserved across a connection.

TIBLCONN — A value greater than zero indicates the maximum number of bytes of user data that can be transferred during connection establishment.

- -1 Indicates that there is no limit on the amount of user data that can be transferred.
- -2 Specifies that the transport provider does not support connect user data.

0 Is not returned by the transport provider.

TIBLDISC

A value greater than zero indicates the maximum number of bytes of user data that can be transferred during connection release.

-1 Indicates that there is no limit on the amount of user data that can be transferred.

-2 Specifies that the transport provider does not support disconnect user data.

0 Is not returned by the transprt provider.

TIBLINFO

A value greater than zero indicates the maximum size of an information unit returned by the TINFO service function for information types other than PRIMARY.

-1 Indicates that there is no limit on the size of an information unit.

-2 Indicates that the transport provider does not support any information types other than PRIMARY.

0 Not returned by the transport provider.

An application program can minimize its dependence on a particular transport provider by using the information defined in the previous tables to determine which facilities and options are supported, and the maximum size of variable-length storage areas used by macro instructions The content and format of the remaining information types is provider-dependent. Only PRIMARY information is supported by the transport providers in the current implementation.

TLISTEN

Listen for a Connect Indication — The TLISTEN macro instruction is used to listen for connect indications generated by connection requests arriving at an endpoint operating in server mode. The endpoint must have been previously enabled with a TBIND macro instruction. The protocol address of the remote transport user that initiated the connection request is returned by the transport provider.

```
[ symbol ] TLISTEN [ EP = endpoint_id ]
                   [ ,ADLEN = protocol_address_length ]
                   [ ,ADBUF = protocol_address_address ]
                   [ ,ADALET = protocol_address_alet ]
                   [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                                [ ,SYNC | ASYNC ]
                                [ ,TRUNC | NOTRUNC ]
                                [ ,BLOCK | NOBLOCK ] ) ]
                   [ ,ECB = INTERNAL | event_control_block_addr ]
                   [ EXIT = tpl_exit_routine_address ]
                   [ ,MF = ( I | L | G | M | E, [ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TLISTEN macro instruction is to be executed.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

ADLEN = *protocol_address_length*

Length (in bytes) of the protocol address storage area identified by the ADBUF operand.

The length is updated when the request is completed to reflect the actual length of the protocol address returned. If the length is zero, the protocol address of the calling transport user is not returned to the application program.

Default: Zero (return no protocol address).

ADBUF = *protocol_address_address*

Address of a storage area for returning the protocol address of the calling transport user.

The storage area should be large enough to contain the entire address. The format of the protocol address is provider-dependent, and its maximum size can be determined by issuing a TINFO macro instruction. The storage area can be aligned on any boundary.

Default: Zero (no protocol address storage area).

ADALET = *protocol_address_alet* Access List Entry Token (ALET) used in access register (AR) mode when referencing the storage specified by the ADBUF parameter.

The ADALET value must be an ALET that is contained in the Dispatchable Unit Access List (DUAL) of the caller. The ADALET parameter may be used only if OPTCD=EXTEND is also specified.

Default: Zero (the storage is contained in the address space of the caller).

OPTCD = SHORT | LONG | EXTEND Format attribute of the parameter list associated with this request.

OPTCD=SHORT a different control block identifier is used to indicate that a subset of the TPL has been generated.

OPTCD=LONG a standard, full-length TPL is generated.

OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix is used to contain ALET address extensions that can be specified by other request parameters.

Default: SHORT if MF=I or MF operand omitted, LONG otherwise.

OPTCD = SYNC | ASYNC Synchronization mode to use when executing the TADDR macro instruction.

OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction is complete. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API.

OPTCD=ASYNC the request executes in asynchronous mode, and returns control immediately after scheduling the TLISTEN request. The application program is responsible for issuing the TCHECK macro instruction.

Default: SYNC (synchronous mode).

OPTCD = TRUNC | NOTRUNC	Indicates whether the information returned to the application program by the transport provider should be truncated if it does not fit within the storage area provided. OPTCD=TRUNC the excess is truncated, and the TLISTEN macro instruction is completed conditionally as long as no other errors occur. OPTCD=NOTRUNC nothing is placed in the storage area, and the TLISTEN macro instruction is completed abnormally. Default: NOTRUNC (no truncation).
OPTCD = BLOCK | NOBLOCK	Indicates whether the issuing task can be suspended if the TLISTEN macro instruction cannot be completed immediately. OPTCD=BLOCK (and no connect indicated is generated) the issuing task is suspended until a connection request arrives. OPTCD=NOBLOCK the macro instruction is completed immediately, and an abnormal return code indicates that the task would have been suspended for an indefinite period of time. The TLISTEN macro instruction can be used to poll for new connect indications. If a connect indication is available, the request is completed as usual. Otherwise, the request is completed abnormally and the transport user can try again after delaying an appropriate period of time. In either case, if a connect indication was generated, the TLISTEN macro instruction completes normally without suspending the issuing task. Default: BLOCK (suspend issuing task if necessary).
ECB = INTERNAL | *event_control_block_addr*	Location of an Event Control Block (ECB) to be posted by the API when the TLISTEN macro instruction associated with this TPL is completed. The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB. The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB. This operand is mutually exclusive with the following EXIT operand. Default: INTERNAL (internal ECB).

EXIT = *tpl_exit_routine_address*

Address of an exit routine to schedule when the TLISTEN macro instruction associated with this TPL is completed.

The TPL exit routine is scheduled only if asynchronous mode has been specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB.

This operand is mutually exclusive with the previous ECB operand.

Default: Not indicated (no TPL exit routine).

MF = (I | L | G | M | E , [*tpl_address*])

Standard, list, generate, modify, or execute form of the TLISTEN macro instruction.

The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used.

See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: MF=I (standard, nonreentrant form).

Completion Information

The TLISTEN macro instruction completes normally (or conditionally) when a connect indication is pending, and the information accompanying the connection request has been returned to the application program. If the number of pending indications is one, the state of the endpoint is changed from enabled (TSENABLD) to connect-indication-pending (TSINCONN). Otherwise, connect indications were already pending, and the state is not changed.

If a storage area was provided by the application program, the protocol address of the calling transport user is returned. The corresponding length of the address buffer storage area is updated to reflect the actual amount of information returned.

A sequence number that identifies the connect indication is returned in the TPLSEQNO field of the TPL associated with this request. If more connect indications have been generated, and are waiting to be received by the application program, TOMORE is set in TPLOPCD2, and the number of available connect indications is returned in the TPLCOUNT field of the TPL.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY), and a conditional completion code is returned in register zero. TCTRUNC is set if the information returned to the application program was truncated to fit in the storage area provided. The TPL return code field is set accordingly. No other information is returned.

If the TLISTEN macro instruction completes abnormally, no information is returned, and the connect indication (if any) remains available. The state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TLISTEN return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY	TCTRUNC	
TRFAILED	TAEXCPTN	TENOBLOK		
	TAINTEG	TEPROTO	TEOVRFLO	TERETRCT
	TAENVIRO	TESYSERR TESTOP	TESUBSYS TETERM	TEDRAIN TEUNSUPF
	TAFORMAT	TEBDFNCD TEBDEXIT	TEBDOPCD TEBDADDR	TEBDECB
	TAPROCED	TEAMODE	TESTATE	TEINCMPL
	TATPLERR	TEACTIVE		
TRFATLFC	func. code	The function code loaded into register zero is invalid.		
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.		

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.

Usage Information

The TLISTEN macro instruction is used to listen for connect indications that are generated at an endpoint. A connect indication is generated when a connection request arrives from a remote transport user requesting a connection to the local endpoint. The indication remains pending until accepted or rejected by the application program, or retracted by the remote transport user.

The TLISTEN macro instruction completes when a connect indication is available. A sequence number is returned in the TPLSEQNO field of the TPL associated with the request to uniquely identify the pending indication. This sequence number must be provided with subsequent macro instructions that accept (TACCEPT) or reject (TREJECT) the connection request. The application program should make no assumptions regarding the format of the sequence number other than it is an unsigned fullword value. In particular, sequential completions of TLISTEN requests do not necessarily generate sequential sequence numbers. In addition, the sequence number is not necessarily a small integer value suitable for array indexing.

A count of the number of unreceived connect indications is returned in the TPLCOUNT field of the TPL. If the value returned is nonzero, another TLISTEN macro instruction should be issued to receive the next connect indication. The API guarantees that a TLISTEN request is completed immediately if the preceding TLISTEN completed with a nonzero indication count. TOMORE is also set in TPLOPCD2 to indicate more connect indications are available. TOMORE corresponds to the option code that is set when OPTCD=MORE is indicated.

The protocol address of the calling transport user is also returned to the application program. The application program can use the protocol address to determine if it should accept or reject the connection request.

The TLISTEN macro instruction is typically used by application programs running in server mode. The protocol address bound to the endpoint is well known by transport users that want to connect to the application program and use its services. Connect indications generated by arriving connection requests are queued, and are presented to the application program as TLISTEN macro instructions are issued. The total number of connect indications that can be queued at one time is determined by the TBIND macro instruction.

The TLISTEN macro instruction does not remove a connect indication from the queue, but only serves to retrieve the information associated with the indication. When a connect indication is received by the application program, it is said to be pending. A TACCEPT or TREJECT macro instruction must be issued to remove the pending indication from the queue. More than one connect indication can be received with TLISTEN macro instructions before any are accepted or rejected, and pending indications can be accepted or rejected in any order.

The number of transport users that can be connected at one time is controlled by the number of endpoints the application program is able to create. However, the number of connection requests that can be awaiting acceptance is controlled by the maximum length of the connect indication queue that was specified when the endpoint was enabled. If this length is zero, the endpoint is disabled, and cannot queue any connect indications. An error is generated if a TLISTEN macro instruction is issued at an endpoint that is disabled.

If OPTCD=BLOCK is indicated when the TLISTEN macro instruction is issued, the request is not completed until a connect indication is available. If the application needs to use the endpoint for some other purpose, an outstanding TLISTEN request must be retracted. The TRETRACT macro instruction causes a pending TLISTEN to complete immediately with a return code indicating the retraction.

The TLISTEN macro instruction is normally issued at endpoints operating in connection mode. However, if an endpoint operating in connectionless mode was enabled for (simulated) connect indications (see TBIND), the TLISTEN macro instruction should be used to receive connect indications generated by arriving datagrams. A subsequent TACCEPT macro instruction creates an association with the remote transport user. See TCPaccess *Assembler API Concepts* for a discussion of associations in connectionless mode.

TOPEN

Open a Transport Endpoint— The TOPEN macro instruction is used to create an endpoint within a given communications domain, and to designate the type of transport service required for the endpoint. Optionally, the TOPEN macro instruction can be used to acquire control of an existing endpoint from another endpoint ID created by another task or in another address space.

```
[ symbol ] TOPEN [ DOMAIN = INET ]
                 [ ,TYPE = ( mode [ ,options ] ) ]
                 [ ,PROTO = protocol_number ]
                 [ ,SVCID = transport_service_id ]
                 [ ,APCB = application_program_control_block_addr ]
                 [ ,EXLST = exit_list_address ]
                 [ ,EVENTLST = event_list_address ]
                 [ ,UCNTX = one_word_of_user_context ]
                 [ ,EP = old_endpoint_id ]
                 [ ,TCB = task_control_block_address ]
                 [ ,ASCB = address_space_control_block_address ]
                 [ ,USER = endpoint_userid ]
                 [ ,MODE = TLI | SOCKETS ]
                 [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                              [ ,SYNC | ASYNC ]
                              [ ,TUB | ACEE ]
                              [ ,PLAIN | CIPHER ]
                              [ ,NEW | OLD ] ) ]
                 [ ,ECB = INTERNAL | event_control_block_addr ]
                 [ ,EXIT = tpl_exit_routine_address ]
                 [ ,MF = ( I | L | G | M | E , [ tspl_address ] ) ]
```

DOMAIN=INET

Communications domain within which the new endpoint exists.

Once a domain is selected and the endpoint created, the domain to which it belongs cannot be changed for the life of the endpoint. If a domain is not specified, one is selected based on the specification of other parameters and installation defaults.

Definition of a domain is independent of the existence of a transport provider on the local system that supports the domain. If a domain is selected for which no transport provider exists, the endpoint is not created. In the current implementation, only a transport provider for DOMAIN=INET is supported

Default: INET.

TYPE = (*mode* [,*options*])

Type of transport service required for the endpoint. The service type in combination with the domain specification is generally sufficient to determine the protocol and transport provider used to provide the service.

If the transport service type is not specified, one is selected based on the specification of other parameters and installation defaults.

The service type is fixed for the life of the endpoint. The service type is a sublist consisting of the mode of service, followed by optional services required by the endpoint.

The mode of service must be one of these, and must be coded as the first sublist operand:

COTS Connection-mode transport service.

CLTS Connectionless-mode transport service.

PROTO=*protocol_number*

Two optional services are defined, and can be requested by specifying one of these keywords:

ORDREL Orderly release required. ORDREL should be requested only with the COTS service mode.

ASSOC Connectionless associations required.

Note: ASSOC should be requested only with the CLTS service mode.

Since the association service is supported entirely within the transport service interface, orderly release is always available when connectionless-mode associations are used.

These combinations are valid types of service:

COTS (COTS,ORDREL)

CLTS (CLTS,ASSOC)

Domains do not necessarily support all transport service types, and for any given service type that is supported by a particular domain, a transport provider may not be available on the local system. If no transport provider exists for the specified service type, the endpoint is not created.

This operand is mutually exclusive with the PROTO operand.

Default: Not indicated (use installation default).

PROTO = *protocol_number*

Protocol number of the transport protocol within a communication domain that is required for the endpoint.

The protocol number in combination with the domain specification is generally sufficient to determine the protocol and transport provider that are used to provide the service. If the protocol number is not specified, one is selected based on the specification of other parameters and installation defaults. The protocol number is fixed for the life of the endpoint.

Generally, the transport service type should be used to designate the required protocol service. However, if more than one candidate protocol exits for a given service type, the specific choice can be indicated with the protocol number. Also, some installations may be able to run production and development versions of the same protocol service, and in this case, a protocol number must be specified to force selection of the development version.

The protocol number is domain-dependent, and identifies a specific protocol.

This operand is mutually exclusive with the TYPE operand.

Default: Not indicated (use installation default).

SVCID = *transport_service_id*

ID of a specific transport service provider. Generally, the transport service type or protocol number is sufficient for selecting the appropriate transport service provider. However, if more than one transport service provider is available that can provide the requested service, the transport service ID must be indicated to select the appropriate provider. The transport service ID is coded as an alphanumeric string up to eight characters in length.

Default: Not indicated (use installation default).

APCB = *application_program_control_block_address*

Address of the APCB that defines the application program and corresponding transport user.

The APCB also defines the MVS subsystem that contains the transport provider. The APCB must be opened by the same task that issued the TOPEN macro instruction. If the APCB is not opened, unpredictable results may occur.

Default: Zero (no APCB address).

EXLST = *exit_list_address*

Links the endpoint with an exit list containing addresses of routines to enter when certain protocol events occur.

This list is created by a TEXLST macro instruction. More than one endpoint can be linked to the same exit list.

If no exit list is provided, the application program is not able to receive immediate notification of asynchronous protocol events. However, notification may still be received synchronously with the completion status that is returned to the application program when a macro instruction completes.

The TPEND asynchronous exit, and the SYNAD and LERAD synchronous exits, are specified in the AOPEN exit list, and cannot be defined in the TOPEN exit list. For more information on exit lists, see the TEXLST macro and TCPaccess *Assembler API Concepts,* which discusses synchronization and exit routines.

Note: This operand is mutually exclusive with the following EVENTLST operand.

Default: Zero (no exit list).

EVENTLST = *event_list_address*

A list of ECBs and exits to use for protocol or shutdown event notification. The function of the event list is identical to the function of the exit list (EXLST parameter), except that an event list supports event notification via ECB posting as well as exit routine execution. The event list is created using the TEVNTLST macro instruction.

Note: This operand is mutually exclusive with the previous EXLST operand.

Default: Zero (no event list).

UCNTX = *one_word_of_user_context*

One arbitrary word of user context to associate with the endpoint.

The information provided is not interpreted by the API, and is merely saved with other endpoint information. It can be retrieved later by the application program, and is useful for getting context within exit routines. This word of context is included in the parameter list passed to any asynchronous exit routine that is entered on behalf of the endpoint.

Default: Zero (no user context).

EP = *old_endpoint_id*

Endpoint being acquired when OPTCD=OLD is specified.

The value specified must be the endpoint ID of an existing endpoint as returned from a TOPEN macro instruction issued by the controlling task or address space. The task relinquishing control of the endpoint must also issue a TCLOSE OPTCD=PASS macro instruction specifying the same endpoint ID.

Default: Zero (create new endpoint).

TCB = *task_control_block_address*

TCB address of the task from which an endpoint is being acquired when OPTCD=OLD is specified.

If the indicated value is zero, the endpoint is acquired from another task in the specified address space that indicates this task's TCB address on a TCLOSE TCB parameter.

If the indicated value is not zero, the value specified must match the TCB address of the task relinquishing control of the endpoint.

Default: Zero (accept from any task).

ASCB = *address_space_control_block_address*

ASCB address of the address space from which an endpoint is being acquired when OPTCD=OLD is specified.

If the indicated value is zero, the endpoint can only be acquired from a task executing within the same address space.

If the indicated value is not zero, the indicated value is the ASCB address of another address space that currently controls the endpoint

The relinquishing address space must issue a TCLOSE macro instruction indicating this address space as the acquirer.

Default: Zero (accept from task within this address space only).

USER = *endpoint_userid*	Associates a user ID with the endpoint for authorization and accounting purposes. OPTCD=TUB the specified value must be the address of a Transport Endpoint User Block (TUB) containing the user information. OPTCD=ACEE the specified value must be the address of an Accessor Environment Element (ACEE) obtained from the local security system when the user ID was authenticated. If the option is not coded, the application name specified in the APCB is used. The password contained in the TUB can be plain text or cipher text depending on the OPTCD=PLAIN|CIPHER operand. If cipher text, it is assumed that the password was encrypted using the encryption mechanism supplied by the local security system. The API merely provides the password to the security system in its encrypted form. The user ID or application name is also supplied to the transport provider. How this information is used is unspecified, and provider-dependent. Default: Zero (no user ID; use application name for accounting and authorization).
MODE = TLI | SOCKETS	Allows TSEND and TSENDTO operate in a mode similar to BSD sockets If the option is MODE=SOCKETS, then data transfer occurs in socket mode. See Usage Notes for TSEND and TSENDTO for additional information on data transfer modes. Default: TLI.

OPTCD = SHORT | LONG | EXTEND	Format attribute of the parameter list associated with this request. OPTCD=SHORT a different control block identifier is used to indicate that a subset of the TPL was generated. OPTCD=LONG a standard, full-lengthTPL is generated. OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters. Default: SHORT if MF=I or MF operand omitted, LONG otherwise.
OPTCD = TUB | ACEE	Format of user ID information referenced by the USER operand. OPTCD=TUB, then user ID, group, and password information are provided in a Transport User Block (TUB). OPTCD=ACEE, the user information is contained in an Accessor Environment Element (ACEE) obtained from the local security system. Default: TUB (user information provided in TUB).
OPTCD = PLAIN | CIPHER	Indicates whether the password contained in the Transport User Block (TUB) designated with the USER operand is encrypted or in plain text form. OPTCD=PLAIN the password is in plain text. OPTCD=CIPHER the password is encrypted. The API uses this information when requesting user ID and password verification from the local security system. Default: PLAIN (password in plain text).

OPTCD = NEW | OLD	Indicates whether a new endpoint should be created, or an existing endpoint should be passed to another task or address space. OPTCD=NEW a new endpoint is to be created, and the EP operand must indicate a zero value. OPTCD=OLD control of an existing endpoint is being acquired from another task or address space. The EP operand indicates the ID of an existing endpoint. Default: NEW (create new endpoint).
ECB = INTERNAL | *event_control_block_addr*	Location of an Event Control Block (ECB) to be posted by the API when the TOPEN macro instruction associated with this TPL completes. The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB. The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB. This operand is mutually exclusive with the following EXIT operand. Default: INTERNAL (internal ECB).
EXIT = *tpl_exit_routine_address*	Address of an exit routine to be scheduled when the TOPEN macro instruction associated with this TPL is completed. The TPL exit routine is scheduled only if asynchronous mode was specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB. This operand is mutually exclusive with the previous ECB operand. Default: Not indicated (no TPL exit routine).

MF = (I | L | G | M | E , [*tpl_address*])

Standard, list, generate, modify, or execute form of the TOPEN macro instruction.

The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used.

Refer to List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: MF=I (standard, nonreentrant form).

Completion Information

The TOPEN macro instruction completes normally when the requested endpoint is created (or acquired), and is ready to be used as the argument of other API macro instructions. The initial state of the endpoint is opened (TSOPENED) if the endpoint is new. If the endpoint is old, and was acquired from another task or address space, the endpoint retains the state that existed when it was closed by the relinquishing task. In this case, the state can be opened (TSOPENED), disabled (TSDSABLD), enabled (TSENABLD), or connected (TSCONNCT).

A token that identifies the endpoint is returned in the TPL as the endpoint ID, and should be used in all subsequent requests that refer to this endpoint. The application program should make no assumptions regarding the format of an endpoint ID, other than it is an unsigned, fullword value. If an existing endpoint was acquired from another task, the TCB and ASCB addresses of the relinquishing task are returned.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY). The conditional completion code in register zero is always zero (TCOKAY), and the TPL return code field is set accordingly. No other information is returned.

If the TOPEN function completes abnormally, the endpoint is not created (or acquired), and no endpoint ID is returned. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TOPEN return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY		
TRFAILED	TAENVIRO	TESYSERR TESTOP TEUNSUPF	TESUBSYS TETERM TEUNAUTH	TEDRAIN TEUNAVBL TERSOURC
	TAFORMAT	TEBDFNCD TEBDEXIT TEBDPROT TEBDUSER TEBDOPCD	TEBDDOM TEBDEPID TEBDACEE TEBDDECB TEBDTYPE	TEBDXLST TEBDTSID TEBADDR TEBDASCB TEBDXECB
	TAPROCED	TEAMODE	TEOWNER	
	TATPLERR	TEACTIVE		
TRFATLFC	func. code	The function code loaded into register zero is invalid.		
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.		
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.		
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.		

Usage Information

The TOPEN macro instruction is used to create a new transport endpoint, or acquire control of an existing endpoint from some other task or address space. An endpoint ID is returned. It is used to identify the newly created (or acquired) endpoint, and must be supplied in all subsequent transport service requests that apply to the endpoint. The endpoint remains opened until closed with a TCLOSE macro instruction.

The endpoint carries with it some context that is fixed for the duration of its use. It belongs to a particular communications domain, and is associated with a particular transport provider active on the local system. The transport provider is the supplier of a transport service using a particular transport protocol with well known characteristics. The domain, transport service and protocol are selected in accordance with the DOMAIN, TYPE, and PROTO operands specified on the TOPEN macro instruction. A service ID can also be specified that selects a particular provider when more than one apply.

The endpoint is linked to the application program via the APCB address that is specified when the endpoint is opened. The APCB also serves to identify the MVS subsystem that contains the transport provider. Once the endpoint is opened, the endpoint ID serves as the anchor for all context related to the endpoint. The APCB must have been opened by the same task that opens the endpoint.

The endpoint can also be associated with a user that is known to the local security system. The user is identified with a Transport User Block (TUB), or if the user ID has already been authenticated by the application program, the address of an ACEE can be supplied. This information is used to determine access authority for the requested service, and is used in subsequent requests to determine access authority for certain resources such as well-known protocol addresses. Any accounting information maintained by the API or the transport provider also contains the user ID.

Throughout the life of the endpoint, several asynchronous protocol events can occur. For example, an endpoint used to listen for connection requests can receive a connect indication, or an endpoint associated with an established connection can suddenly become disconnected. An exit list, generated by the TEXLST or TEVNTLST macro instruction, is used to designate exit routines for handling asynchronous events. The address of the exit list is specified by the EXLST or EVENTLST operand.

The exit list is linked to the endpoint at the time it is created, and if no exit list is specified, the exit list linked to the APCB is used in its place. If no exit list exists, or a particular protocol exit has not been enabled, the corresponding event must be processed synchronously.

An endpoint can only be closed by the task that opened it. The opening task is said to control (or own) the endpoint, although other tasks may issue macro instructions that reference the endpoint. If it is necessary for another task to close an endpoint, control must be acquired by the closing task. The current owner passes control by closing the endpoint with OPTCD=PASS indicated. The new owner acquires control by opening the endpoint with OPTCD=OLD indicated. Ownership can be passed to a task in another address space. The task and address space are identified by the TCB and ASCB operands.

When control of an endpoint is passed to another address space, local endpoint context must be recreated in the new address space. In addition, since local storage used to maintain the context is associated with the task that allocates it, this context must be recreated, even when passing control to another task within the same address space. Therefore, even though a passed endpoint retains most of its existing context, a new endpoint ID is assigned. The application program must use the endpoint ID returned by TOPEN in all future service requests, and the old endpoint ID should be discarded. Neither the acquiring or relinquishing tasks should ever reference the old endpoint ID once the TOPEN and TCLOSE macro instructions have completed.

TOPTION

Manage Options for Transport Endpoint—Protocol options associated with an endpoint are managed using the TOPTION macro instruction. Options can be declared, queried or verified, and default options used by the transport provider can be retrieved.

```
[ symbol ] TOPTION [ EP = endpoint_id ]
                   [ ,OPLEN = protocol_options_length ]
                   [ ,OPBUF = protocol_options_address ]
                   [ ,OPALET = protocol_options_alet ]
                   [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                                [ ,SYNC | ASYNC ]
                                [ ,NEGOT | NONEGOT ]
                                [ ,DECLARE | VERIFY | QUERY | DEFAULT ]
                                [ ,TP | API ] ) ]
                   [ ,ECB = INTERNAL | event_control_block_addr ]
                   [ ,EXIT = tpl_exit_routine_address ]
                   [ ,MF = ( I | L | G | M | E , [ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TOPTION macro instruction will execute.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

OPLEN = *protocol_options_length*

Length (in bytes) of the protocol option list identified by the OPBUF operand.

A value of zero indicates there is no protocol option list, and is invalid for the TOPTION macro instruction.

Default: Zero (no protocol option list).

OPBUF = *protocol_options_address*

Address of a storage area containing a protocol option list.

The area must contain a list of variable-length protocol options, with each option identified by its length and name. Each entry in the list must also contain room for an option value, which is initialized with the desired value of the option for the DECLARE and VERIFY forms of the TOPTION macro instruction. For the DEFAULT and QUERY forms, the option value is returned in the storage area provided.

The type, number and format of protocol options are provider-dependent, and the maximum size of the option list can be determined by issuing a TINFO macro instruction. The storage area can be aligned on any boundary convenient to the application program.

Default: Zero (no protocol option list).

OPALET = *protocol_options_alet*

Access List Entry Token (ALET) used in access register (AR) mode when referencing the storage specified by the OPBUF parameter.

The OPALET value must be an ALET contained in the Dispatchable Unit Access List (DUAL) of the caller. The OPALET parameter can be used only if OPTCD=EXTEND is also specified.

Default: Zero (the storage is contained in the address space of the caller).

OPTCD = SHORT | LONG | EXTEND	Format attribute of the parameter list associated with this request. OPTCD=SHORT a different control block identifier is used to indicate that a subset of the TPL was generated. OPTCD=LONG a standard, full-length TPL is generated. OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters. Default: SHORT if MF=I or MF operand omitted, LONG otherwise.
OPTCD = SYNC | ASYNC	Synchronization mode to use when executing the TOPTION macro instruction. OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction completes. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API. OPTCD=ASYNC the request executes in asynchronous mode, and returns control immediately after scheduling the TOPTION request. The application program is responsible for issuing the TCHECK macro instruction. Default: SYNC (synchronous mode).
OPTCD = NEGOT | NONEGOT	Indicates whether protocol options associated with this request can be negotiated to an inferior value. OPTCD=NEGOT protocol options are negotiated to comply with the limits of the transport provider, and the conditional completion code is set to indicate that the negotiation occurred. OPTCD=NONEGOT negotiation is disallowed, and unacceptable options result in the abnormal completion of the TOPTION macro instruction. Default: NONEGOT (negotiation disallowed).

OPTCD = DECLARE | VERIFY | QUERY | DEFAULT	The action to be performed by the TOPTION macro instruction. Protocol options that are the subject of the actions listed in this table are contained or returned in an option list designated by the OPLEN and OPBUF operands. One of these actions may be indicated: OPTCD=DECLARE the options specified by the application program are invoked, and the option list is updated with the inferior value of any negotiated options. OPTCD=VERIFY the options specified by the application program are verified, and the option list is updated with the inferior value of any negotiated options. OPTCD=QUERY the current value of options selected by the application program are returned. OPTCD=DEFAULT the default value of options selected by the application program are returned. The type, number and format of protocol options supported by a transport provider are protocol specific. Default: DECLARE (invoke protocol options).
OPTCD = TP | API	Indicates whether the option list identified by the OPBUF and OPLEN operands contains transport interface or transport provider options. OPTCD=API the option list contains interface options that are processed solely by the API. OPTCD=TP the option list is passed to the transport provider for processing. Transport interface and transport provider options can only be manipulated with separate invocations of the TOPTION macro instruction. Default: TP (transport provider options).

ECB = INTERNAL | *event_control_block_addr*	Location of an Event Control Block (ECB) to be posted by the API when the TOPTION macro instruction associated with this TPL completes. The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field used to store the ECB address is used as an internal ECB. The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB. This operand is mutually exclusive with the following EXIT operand. Default: INTERNAL (internal ECB).
EXIT = *tpl_exit_routine_address*	Address of an exit routine to be scheduled when the TOPTION macro instruction associated with this TPL is completed. The TPL exit routine is scheduled only if asynchronous was specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB. This operand is mutually exclusive with the previous ECB operand. Default: Not indicated (no TPL exit routine).
MF = (I | L | G | M | E , [*tpl_address*])	Standard, list, generate, modify, or execute form of the TOPTION macro instruction. The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used. See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters. Default: MF=I (standard, nonreentrant form).

Completion Information

The TOPTION macro instruction completes normally (or conditionally) when the protocol options specified by the application program have been processed by the transport provider. If OPTCD=DECLARE was specified, the indicated options have been negotiated and set to the requested values. Negotiated, verified, or queried values of the indicated options are returned in the storage area provided by the application program. The state of the endpoint is not changed.

On return to the application program, the general return code in register 15 is set to zero (TROKAY), and a conditional completion code is returned in register zero. TCNEGOT is set in the conditional completion code if any of the protocol options specified by the application program were negotiated to an inferior value, and OPTCD=DECLARE was indicated. TCVERIFY is set if any of the options specified by the application program are not supported by the transport provider, and OPTCD=VERIFY was indicated. The TPL return code field is set accordingly. No other information is returned.

If the TOPTION macro instruction completes abnormally, no protocol options are negotiated, set, or returned. All options in effect at the time the TOPTION macro instruction was executed remain in effect. The state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TOPTION return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/ Explanation		
TROKAY	TAOKAY	TCNEGOT	TCVERIFY	
TRFAILED	TAEXCPTN	TENONEGO		
	TAENVIRO	TESYSERR TESTOP TEUNAUTH	TESUBSYS TETERM TERSOURC	TEDRAIN TEUNSUPO
	TAFORMAT	TEBDFNCD TEBDEXIT	TEBDOPCD TEBDOPTN	TEBDECB
	TAPROCED	TEAMODE	TESTATE	TEINCMPL
	TATPLERR	TEACTIVE		
TRFATLFC	func. code	The function code loaded into register zero is invalid.		
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.		
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.		
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.		

Usage Information

The TOPTION macro instruction is used to manage protocol options associated with an endpoint. Protocol options can be specified by the application program, and options the application program intends to specify can first be verified to determine if they are supported by the transport provider. Default options that are in effect if not overridden by the application program can be retrieved, and the current value of options in effect for the endpoint can be queried. The action to be taken by the TOPTION macro instruction is indicated by the OPTCD operand.

Multiple options can be manipulated with a single invocation of the TOPTION macro instruction. The options are identified by an option list provided by the application program, and updated by the transport provider. Each entry in the list represents one option, and contains the length and name of the option. The format of the option name is provider-dependent, but usually identifies a protocol level and option supported by the protocol. Each entry also contains room for an option value, whose length is option-dependent. The format of an option list entry for a specific transport provider is defined by the TPO DSECT (see TDSECT).

The following table shows the general format:

TOPTION Format of Option Name

x+0	OPTION LENGTH	OPTION NAME
x+4	OPTION VALUE	

x+optlen

The type, number, and format of protocol options supported by the transport provider is protocol and provider dependent. The application should be conservative in its use of protocol options to remain independent of a specific transport provider. The options supported by specific transport providers, and the format of their specification, are documented in the provider-specific appendix at the end of this reference.

Although a transport provider supports a particular option, there is no guarantee it can support the value requested at the time of the request. If necessary, the transport provider negotiates the option to an inferior value in order to successfully complete the macro instruction.

- If OPTCD=NONEGOT is specified, no negotiation is allowed, and the macro instruction is completed abnormally.
- If OPTCD=NONEGOT is not specified, the macro instruction is completed conditionally, and the fact that an option was negotiated is indicated by the conditional completion code. The option list is updated with the negotiated option value.

Transport Provider Options

The TOPTION macro instruction is generally used to manipulate transport provider options. However, if so indicated by the OPTCD operand, the TOPTION macro instruction can also be used to manipulate transport interface options. Transport interface (OPTCD=API) and transport provider (OPTCD=TP) options cannot be combined in the same options list. The API option names are defined by the TPO DSECT. These names correspond to the symbols used in the DSECT to define the option name.

These names describe OPTCD=API options. The values are all four bytes long so the option length must be eight bytes.

TPOAQSND	The maximum number of TSEND or TSENDTO macro instructions that can be executed at an endpoint without waiting for at least one to complete. In other words, TPOAQSND is the maximum number of pending send requests.
TPOAQRCV	The maximum number of TRECV or TRECVFR macro instructions that can be executed at an endpoint without waiting for at least one to complete. In other words, TPOAQRCV is the maximum number of pending receive requests.
TPOALSND	The maximum number of bytes of data that can be pending for outstanding TSEND or TSENDTO requests. In other words, TPOALSND is the total amount of send buffering allocated for an endpoint.
TPOALRCV	The maximum number of bytes of data that can be pending for outstanding TRECV or TRECVFR requests. In other words, TPOALRCV is the total amount of receive buffering allocated for an endpoint.

TCP Provider Session Options

These options are valid only for TCP provider sessions. These names describe OPTCD=TP options. The values are all four bytes long, so the option length must be eight.

TPOPRWND

The size of the receive buffer used by TCP. This is reflected as the receive window advertised by TCP. This option is valid only for TCP sessions and must be set before a connection is established. The range of acceptable values is 256-500,000.

Values above 65535 are valid and allocate a buffer of the specified size. A sliding window of 65535 is used in this buffer by TCP.

Default: 261376.

TPOPKTIM

The interval of idle time used by the keepalive option specified in minutes. The range of acceptable values is 1-1439.

Default: 120.

TPOPKEEP

The keepalive option enables the periodic probing of the remote. This option specifies the type of keepalive to use. It is also used to turn off the keepalive option. These integer values are supported:

0 Turn off keepalive.

1 Use keepalive with no data, and do not abort the session if no response.

2 Use keepalive with no data, and abort the session if no response.

3 Use keepalive with data.

See the notes following this table for a discussion of keepalive. Their use is discouraged and is provided only for those applications that are incapable of detecting idle sessions.

TPOPDNAG

Defeat TCP's Nagle algorithm used to gather send data into maximal packets. A value of one defeats the Nagle algorithm.

A value of zero restores normal operation of the Nagle algorithm. This option is only valid when a connection is established. It is intended for use by applications that send small amounts of data and for performance reasons demand that they be sent in small packets.

High volume applications should not use this option and should use multiple asynchronous TSENDs instead.

Note: These notes apply to the keepalive option (that is, TPOPKEEP):

The use of keepalive is discouraged by the internet community. It should be the responsibility of the application to detect idle connections and probe them or terminate them as appropriate. However, here is a brief description of their use:

- The keepalive options may only be set when a connection is established. Once set, they do not become effective until there is network activity, either a TSEND or a TRECV. This also applies to turning keepalive off.
- Keepalive packets may be sent with or without data. Normally they are sent without data. Since some implementations do not respond to keepalive of this form, excessive retransmissions of the keepalive does not abort the session. However, if the session has terminated at the remote end, that host sends a reset, aborting the connection. If the host does not respond at all, you can request that the session be aborted after excessive retransmissions.
- If it is determined that the remote host implementation does not respond to a keepalive with no data, you can request that keepalive be sent with one byte of data. The retransmission mechanism aborts the session if retransmissions are exceeded.

See RFC 1122 for a more complete description of keepalive considerations.

TCP/UDP/RAW Provider Session Options

These options are derived from the Berkeley socket implementation and are valid for TCP, UDP or RAW sessions. These options are used with OPTCD=TP options.

TPOIPOPT Set or get options for IP protocol

The maximum total length of IP options is 40 bytes.

Most IP protocol options are itemized in standard IP header format starting with:

- Option Code: 1 byte
- Length of segment: 1 byte
- Pointer to first variable data field: 1 byte
- Variable length data fields

Supported options are:

0 End-of-options (This option is a single byte and does not use the standard IP option format.)

1 No-op (This option is a single byte and does not use the standard IP option format.)

7 Record Route Option

Code	Length	Pointer	4-byte IP Address1	.....	4-byte IP Address9
7	7-39	4			

68 Timestamp Option

The Timestamp Option uses two different formats depending on the operation requested within the Timestamp Option Flags field.

Code	Length	Pointer	Overflow	Flags	4-byte IP Address1	4-byte Timestamp #1	.....	4-byte IP Address4	4-byte Timestamp #4
68	12-36	5							

Code	Length	Pointer	Overflow	Flags	4-byte Timestamp #1	.....	4-byte Timestamp #9
68	8-40	5					

Note: Overflow and Flags fields are four bits each.

Overflow: count of additional hops not timestamped

Flags:

0 = Record timestamps only

1 = Record IP address/timestamp pairs

2 = Record timestamps for pre-set IP addresses

131 Loose Source and Record Route Option

Code	Length	Pointer	4-byte IP Address1	.....	4-byte IP Address9	4-byte IP Destination Address
131	11-43	4				

Route through the specified addresses; additional hops may be taken. The last address in the list must be the final destination.

137 Strict Source and Record Route Option

Code	Length	Pointer	4-byte IP Address1	.....	4-byte IP Address9	4-byte IP Destination Address
137	11-43	4				

Route only through each address as specified. The last address in the list must be the final destination.

TPOSIOAR Add a single routing table entry. This socket option is for internal use only.

The option is failed when requested from outside the transport provider address space. This option requires an 88-byte field to specify the route data. The route data is mapped by T01DIRT (internal DSECT macro).

TPOSIODR Delete a single routing table entry. This socket option is for internal use only.

The option is failed when requested from outside the transport provider address space. This option requires an 88-byte field to specify the route data. The route data is mapped by T01DIRT (internal DSECT macro).

TPOIFNO Get the number of interfaces. Provide a four-byte buffer for return of the count.

TPOSIFCF Get the interface configuration list. The list is returned in a buffer which is sized by (number of interfaces * 32) + 4, where the first four bytes contain the length of the configuration list. Fields returned for each interface are:

Offset	Length	Description
0	16	Name of the interface
16	16	Interface network address

The first eight bytes of the interface network address can be mapped using the Transport Protocol Address (TPA) DSECT. The last eight bytes are padding.

TPOSIFLG Get interface flags. Provide an 18-byte buffer with the interface name specified in the first 16 bytes. Associated flags are returned in bytes 17-18.

See data structure TIOC for flag definitions.

TPOSIFMT Get maximum transmission unit. Provide a 20-byte buffer with the interface name specified in the first 16 bytes. The associated MTU is returned in bytes 17-20.

TPOSIFME Get metric. Provide a 20-byte buffer with the interface name specified in the first 16 bytes. The associated metric is returned in bytes 17-20.

TPOSIFNM Get network address mask. Provide a 32-byte buffer with the interface name specified in the first 16 bytes. The associated network mask is returned in bytes 17-24.

This information can be mapped using the Transport Protocol Address (TPA) DSECT (for example, the address mask is at offset +4 within this structure).

Bytes 25-32 are padding.

TPOSIFBA Get the broadcast address. Provide a 32-byte buffer with the interface name specified in the first 16 bytes. The associated address is returned in bytes 17-24.

This information can be mapped using the Transport Protocol Address (TPA) DSECT (for example, the broadcast address is at offset +4 within this structure).

Bytes 25-32 are padding.

TPOSIFAD Get the interface address. Provide a 32-byte buffer with the interface name specified in the first 16 bytes. The associated address is returned in bytes 17-24.

This information can be mapped using the Transport Protocol Address (TPA) DSECT (for example, the interface address is at offset +4 within this structure). Bytes 25-32 are padding.

TPOSIFEN Get the hardware address. Provide a 32-byte buffer with the interface name specified in the first 16 bytes.

The associated address is returned beginning at byte 17 and extending for a length that is dependent on the link layer addressing that is in use (for example, the length would be six in the case of Ethernet). The remainder of the buffer is padding.

TPOSIFDS Get the destination address. Provide a 32-byte buffer with the interface name specified in the first 16 bytes. The associated address is returned in bytes 17-24.

This information can be mapped using the Transport Protocol Address (TPA) DSECT (for example, the destination address is at offset +4 within this structure). Bytes 25-32 are padding.

TPOIPTTL Set or get IP time-to-live.

Provide a four-byte field for the maximum number of routing hops to be taken in a range from 1 to 255.

TPOIPTOS Set or get type of service. Provide a four-byte field for the type of service in a range from zero to 255.

TPOTPMSS Set or get maximum segment size. Provide a four-byte field for the maximum segment size in a range from 512 to 64 KB -20.

The number may reflect the maximum transmission unit size less the size of the IP header (20 bytes).

TPOIPDNR Set or get IP do-not-route. Provide a four-byte field.

A value of one indicates that data not be sent through any router, and is restricted to destinations on the local network.

TPOIPBRO Set or get IP broadcast. Provide a four-byte field.

A value of one indicates that broadcasting is allowed.

TPOUDSUM Set or get UDP checksums option. Provide a four-byte field.

A value of one indicates that UDP checksumming is in effect.

TPOREUSE Set or get reuse address option. Provide a four-byte field.

A value of one indicates for TBIND to use a server port number even though the port number is in use by another server.

TPOUDATA Set user data. Provide a 24-byte field with character data to be displayed by NETSTAT.

The 24-byte field breaks down as follows:

Bytes	Value
0-7	User ID
8-15	Secondary Logical Unit (SLU)
16-23	Primary Logical Unit (PLU) or service

Unicenter TCPaccess does not verify the contents of the data, it simply accepts it and assumes that the TU invoking TOPTION is correct.

The TOPTION macro instruction can be issued when the endpoint is in the opened (TSOPENED), disabled (TSDSABLD), enabled (TSENABLD), and connected (TSCONNCT) states. However, for COTS endpoints, some protocol options may not be changed after the endpoint is connected. In the future, some protocol options may also be specified with the TCONNECT and TACCEPT macro instructions. Similarly, in the future, for CLTS endpoints operating in pure datagram mode (that is, without associations), options may be specified with each datagram sent.

TPL

Create a Transport Service Parameter List— The TPL macro instruction is used to create a Transport Service Parameter List (TPL), which is the primary argument of all transport service functions.

```
[ symbol ] TPL [ EP = endpoint_id ]

[ ,ADLEN = protocol_address_length ]
         [ ,ADBUF = protocol_address_address ]
               [ ,ADALET = protocol_address_alet ]
               [ ,DALEN = user_data_length ]
               [ ,DABUF = user_data_address ]
               [ ,DAALET = user_data_alet ]
               [ ,OPLEN = protocol_options_length ]
               [ ,OPBUF = protocol_options_address ]
               [ ,OPALET = protocol_options_alet ]
               [ ,QLSTN = listen_queue_length ]
               [ ,NEWEP = new_endpoint_id ]
               [ ,SEQNO = sequence_number ]
               [ ,USER = endpoint_userid ]
               [ ,TCB = task_control_block_address ]
               [ ,ASCB = address_space_control_blk_addr ]
               [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                            [ ,SYNC | ASYNC ]
                            [ ,TRUNC | NOTRUNC ]
                            [ ,NEGOT | NONEGOT ]
                            [ ,BLOCK | NOBLOCK ]
                            [ ,ASSIGN | USE ]
                            [ ,LOCAL | REMOTE ]
                            [ ,PRIMARY | SECNDRY | STATS ]
                 [ ,DECLARE | VERIFY | QUERY | DEFAULT ]
                            [ ,TP | API ]
                            [ ,MORE | NOMORE ]
                           [ ,NORMAL | EXPEDITE ]
                            [ ,EOM | NOTEOM ]
                            [ ,DIRECT | INDIR ]
                            [ ,ABORT | CLEAR ]
                            [ ,DELETE | PASS ]
                            [ ,TUB | ACEE ]
                            [ ,PLAIN | CIPHER ] ) ]
                            [ ,NOFULL | FULL ]
                            [ ,NOTIMEOUT | TIMEOUT ]
                            [ ,MBUF | NOMBUF ] ) ]
               [ ,FNCCD = TACCEPT | TADDR | TBIND | TCLEAR |
                          TCLOSE | TCONFIRM | TCONNECT |
                          TDISCONN | TINFO | TLISTEN | TOPTION |
                          TRECV | TRECVERR | TRECVFR |
                          TREJECT | TRELACK | TRELEASE | TRETRACT |
                          TSEND | TSENDTO | TUNBIND | TUSER ]
               [ ,ECB = INTERNAL | event_control_block_addr ]
               [ ,EXIT = tpl_exit_routine_address ]
               [ ,MF = ( L | M , [ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint associated with the TPL. When the TPL is later executed using a TEXEC or other API macro instruction, the requested function is performed on the indicated endpoint.

The value specified must be a valid endpoint ID returned from a TOPEN macro instruction.

Default: Zero (no endpoint specified).

ADLEN = *protocol_address_length*

Length (in bytes) of the protocol address storage area identified by the ADBUF operand.

If the storage area contains a protocol address to be supplied to the transport provider, the length indicated should be the actual length of the protocol address.

If the storage area is to be used by the transport provider for returning a protocol address, the length indicated should be the maximum length of the storage area. The transport provider updates the ADLEN field in the TPL to indicate how many bytes were returned.

A length of zero can be specified indicating there is no protocol address in the storage area, or one is not to be returned by the transport provider, depending on the semantics of the request. If the length is nonzero, ADBUF must be coded and indicate a valid storage area.

Default: Zero (no protocol address).

ADBUF = *protocol_address_address*

Address of a protocol address storage area whose length is specified by the ADLEN operand.

If the semantics of the request require a protocol address to be supplied to the transport provider. the storage area must contain a valid protocol address.

If the semantics of the request require the transport provider to return a protocol address, the storage area is updated, and should be large enough to contain the protocol address. The maximum size of a protocol address may be obtained from the transport provider with the TINFO macro instruction.

An address of zero can be specified indicating there is no protocol address to be supplied or returned. However, if the address is zero, the length must also be zero.

There are no alignment restrictions for this storage area. The length of a protocol address storage area is variable to accommodate a variety of transport providers, or to accommodate transport protocols that use variable length protocol addresses. The structure and content of a protocol address is provider-dependent. Information for specific transport providers can be found in the appropriate appendix at the end of this manual.

Default: Zero (no protocol address storage area).

ADALET = *protocol_address_alet* Access List Entry Token (ALET) that is used in access register (AR) mode when referencing the storage specified by the ADBUF parameter.

The ADALET value must be an ALET that is contained in the Dispatchable Unit Access List (DUAL) of the caller. The ADALET parameter may be used only if OPTCD=EXTEND is also specified.

Default: Zero (the storage is contained in the address space of the caller).

DALEN = *user_data_length* Length (in bytes) of the data storage area identified by the DABUF operand.

If the storage area contains data to be supplied to the transport provider (for example, a send request), the length indicates that it should be the actual length of the data.

If the storage area is to be used by the transport provider for returning data (for example, a receive request), the length indicates that it should be the maximum length of the storage area. The transport provider updates the DALEN field in the TPL to indicate how many bytes were returned.

A length of zero can be specified indicating there is no data in the storage area, or none is to be returned by the transport provider, depending on the semantics of the request. If the length is nonzero, DABUF must be coded and indicate a valid storage area. Specifying a length of zero may be invalid for certain types of requests, and if so, generates an error.

Default: Zero (no user data).

DABUF = *user_data_address*

Address of a data storage area whose length is specified by the DALEN operand.

If the semantics of the request require data to be supplied to the transport provider, the storage area must contain the required user data.

If the semantics of the request require the transport provider to return data, the storage area is updated, and should be large enough to contain the desired amount of data.

An address of zero can be specified indicating there is no data to be supplied or returned. However, if the address is zero, the length must also be zero. There are no alignment restrictions for this storage area.

The content of the storage area varies depending on the type of request (for example, if the function requested is TCONNECT, the data storage area contains connect user data). Similarly, if the function is TDISCONN, the data storage area contains disconnect user data. For TRECV and TSEND functions, the storage area contains arbitrary application data. A given transport provider may not support all data types. The API does not interpret the content of any data contained in the storage area.

Just as the content of the data storage area varies depending on the transport function requested, so does the maximum length. The maximum size of various data types may be obtained from the transport provider with the TINFO macro instruction.

Default: Zero (no user data storage area)

DAALET = *user_data_alet*

Access List Entry Token (ALET) used in access register (AR) mode when referencing the storage specified by the DABUF parameter.

The DAALET value must be an ALET that is contained in the Dispatchable Unit Access List (DUAL) of the caller. The DAALET parameter can be used only if OPTCD=EXTEND is also specified.

Default: Zero (the storage is contained in the address space of the caller).

OPLEN = *protocol_options_length* Length (in bytes) of the protocol options storage area identified by the OPBUF operand.

If the storage area contains protocol options to be supplied to the transport provider, the length indicates that it should be the actual length of the protocol options.

If the storage area is to be used by the transport provider for returning protocol options, the length indicates that it should be the maximum length of the storage area. The transport provider updates the OPLEN field in the TPL to indicate how many bytes were returned.

A length of zero can be specified indicating there are no protocol options in the storage area, or none are to be returned by the transport provider, depending on the semantics of the request. If the length is nonzero, OPBUF must be coded, and indicate a valid storage area.

Default: Zero (no protocol options).

OPBUF = *protocol_options_address* Address of a protocol options storage area whose length is specified by the OPLEN operand.

If the semantics of the request require protocol options to be supplied to the transport provider, the storage area must contain valid protocol options.

If the semantics of the request require the transport provider to return protocol options, the storage area is updated, and should be large enough to contain the protocol options. The maximum size of protocol options may be obtained from the transport provider with the TINFO macro instruction.

An address of zero can be specified indicating there are no protocol options to be supplied or returned. However, if the address is zero, the length must also be zero. There are no alignment restrictions for this storage area.

The length of a protocol options storage area is variable to accommodate a variety of transport providers, or to accommodate transport protocols that use variety of protocol options. The type, number and format of protocol options is provider-dependent. Information for specific transport providers can be found in the appropriate appendix at the end of this manual.

Default: Zero (no protocol options storage area).

OPALET = *protocol_options_alet*

Access List Entry Token (ALET) that is used in access register (AR) mode when referencing the storage specified by the OPBUF parameter.

The OPALET value must be an ALET that is contained in the Dispatchable Unit Access List (DUAL) of the caller. The OPALET parameter may be used only if OPTCD=EXTEND is also specified.

Default: Zero (the storage is contained in the address space of the caller).

QLSTN = *listen_queue_length*

Size of the queue for holding incoming connections arriving at an endpoint, and pending connect indications received by the application program.

If the value specified is zero, no connections can be queued, and the endpoint is disabled for receiving connections.

If the value specified is nonzero, incoming connections are queued, and corresponding connect indications are generated at the endpoint.

A connect indication remains pending until it is accepted or rejected by the application program, or until the connection is abandoned by the caller. The value of this operand generally determines whether the application program is operating in client or server mode.

The transport provider may not be able to queue the number of connections specified by the application program, and as a result, attempts to negotiate the indicated value to a lesser amount. If negotiation is permitted by the application program (OPTCD=NEGOT), the request completes conditionally, and returns a conditional completion code in register zero. Otherwise, the TBIND request completes abnormally.

Default: Zero (endpoint is disabled).

NEWEP = *new_endpoint_id*

Endpoint at which a connection is established when a TACCEPT function is executed at an endpoint with a pending connect indication.

The SEQNO operand specifies which connect indication is being accepted. The value specified for NEWEP is the endpoint ID returned from a TOPEN macro instruction. A value of zero can be used to indicate the listening endpoint.

The connected endpoint may be the endpoint at which the connect indication arrived, or a new endpoint.

If NEWP specifies a new endpoint, the endpoint must be in the disabled (TSDSABLD) state. A new endpoint must have a local protocol address bound to it before a connect indication can be accepted. The local protocol address may be the same as that bound to the listening endpoint, or different (if supported by the underlying protocol).

If NEWP specifies an existing endpoint, NEWP is the endpoint ID of the listening endpoint (or zero). The endpoint must not have any pending connect indications other than the one being accepted (that is, the endpoint must be in the connect-indicating-pending state (TSINCONN), and the number of queued indications must be one).

Default: Zero (connection established at listening endpoint).

SEQNO = *sequence_number*

Used by the TACCEPT or TREJECT macro instructions to specify which of several potential pending connect indications is to be accepted or rejected.

The specified value is the *sequence_number* that was returned by the transport provider with the completion of a TLISTEN macro instruction, previously executed at the same endpoint.

Default: Zero (most likely an invalid sequence number).

USER = *endpoint_userid*	Associates a user ID with the endpoint for authorization and accounting purposes. OPTCD=TUB the specified value must be the address of a Transport Endpoint User Block (TUB) containing the user information. OPTCD=ACEE the specified value must be the address of an Accessor Environment Element (ACEE) obtained from the local security system when the user ID was authenticated. If the option is not coded, the application name specified in the APCB is used. The password contained in the TUB may be plain text or cipher text depending on the OPTCD=PLAIN | CIPHER operand. If cipher text, it is assumed that the password was encrypted using the encryption mechanism supplied by the local security system. The API merely provides the password to the security system in its encrypted form. The user ID or application name is also supplied to the transport provider. How this information is used is unspecified, and provider-dependent. Default: Zero (no user ID; use application name for accounting and authorization).
TCB = *task_control_block_address*	Used by the TCLOSE macro instruction to specify the TCB address of a task that is to receive control of an endpoint when closed with OPTCD=PASS. A value of zero causes the endpoint to be passed to any task that issues a complementary TOPEN macro instruction. Default: Zero (pass to any task).
ASCB = *address_space_control_blk_addr*	TCLOSE macro instruction uses this option to specify the ASCB address of an address space that is to receive control of an endpoint when closed with OPTCD=PASS. A value of zero is taken to mean the current address space. Default: Zero (pass to current address space).

OPTCD = SHORT | LONG | EXTEND

Format attribute of the parameter list associated with this request.

OPTCD=SHORT a different control block identifier is used to indicate that a subset of the TPL has been generated.

OPTCD=LONG a standard, full-length TPL is generated.

OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters.

Default: SHORT if MF=I or MF operand omitted, LONG otherwise.

OPTCD = SYNC | ASYNC

When a request is issued in asynchronous mode, control is immediately returned to the application program after the API accepts the request.

When the requested function completes, the API does one of these actions:

- If an ECB was specified, (either internal or external), the API posts a completion indicator in the event control block. The application program must issue a TCHECK or system WAIT macro instruction to determine whether the ECB has been posted. If a system WAIT or similar technique is used, the application program must still issue a TCHECK macro instruction to mark the TPL inactive, and to cause entry to the SYNAD or LERAD exit routine if the request completes with an error.
- If the EXIT operand is in effect for the TPL, the API schedules the exit routine indicated by this operand. The TPL exit routine should issue the TCHECK macro instruction to set the TPL inactive, and to cause entry into a SYNAD or LERAD exit routine if the requested function completed with an error. TCHECK must be issued even if a SYNAD or LERAD exit routine was not provided.

Note: For more information on task synchronization, see the TCPaccess Assembler API Concepts.

The application program should not modify an active TPL. A TPL is considered active from the time a request is accepted until it is marked inactive by the TCHECK macro instruction. Modifying an active TPL yields unpredictable results, and may cause the request or application program to terminate abnormally.

The coding of the ECB and EXIT operands must be consistent with the OPTCD operand. For example, if OPTCD=SYNC is indicated, the ECB and EXIT operands should not be coded. However, since synchronous mode uses an internal ECB, ECB=INTERNAL is permitted when OPTCD=SYNC is indicated.

If an external ECB or exit routine is specified, the default synchronization mode is changed to asynchronous. If OPTCD=SYNC is indicated in this case, an error message is generated at assembly time.

Default: SYNC (synchronous mode).

OPTCD = TRUNC | NOTRUNC

Indicates whether items being returned to the application program by the transport provider are truncated if they do not fit within the storage area provided.

These actions are taken in accordance with the setting of this option:

OPTCD=TRUNC the data or information being returned is truncated at the end of the storage area, and the request is completed conditionally as long as no other errors occur. The length of the storage area as provided by the application program remains unchanged, and the residual data is discarded. TCTRUNC is set in the conditional completion code.

Note: If OPTCD=TRUNC is specified, and a transport function completes conditionally with TCTRUNC set, some significant data may have been lost. The application programmer should be cautious when invoking this option, and should be fully aware of what data or information might be discarded. The TINFO macro instruction can be used to determine the maximum lengths for various data types supported by the transport provider.

OPTCD=NOTRUNC no data or information is written into the storage area, and the request is completed abnormally. The storage area length is not changed, and there is no way for the application program to determine how much of a deficit there was.

Default: NOTRUNC (truncation disallowed).

OPTCD = NEGOT | NONEGOT

Indicates whether protocol options associated with a request can be negotiated to an inferior value if the transport provider cannot support the option as specified.

These actions are taken in accordance with the setting of this option:

OPTCD=NEGOT the transport provider is free to negotiate an option to an inferior value in order to complete the requested function. If no other errors occur, the request is completed conditionally. TCNEGOT is set in the conditional completion code. In some cases, the negotiated value is returned to the application program at the completion of the request, and in others, an additional macro instruction must be issued to retrieve the negotiated value.

Note: This option provides a degree of independence from the transport provider and the particular protocol in use.

OPTCD=NONEGOT the transport provider may not negotiate an option to an inferior value. If the transport provider cannot support the option as indicated, the request is completed abnormally.

If the application requested a certain quality of service, a lower quality of service may work nearly as well, albeit with diminished performance.

Portability of an application is enhanced when the dependence on protocol options is minimized.

Default: NoneGOT (negotiation disallowed).

OPTCD = BLOCK | NOBLOCK

Indicates whether the issuing task can be suspended if the TPL macro instruction cannot be completed immediately.

OPTCD=BLOCK (and no connect indicated was generated) the issuing task is suspended until a connection request arrives.

OPTCD=NOBLOCK the macro instruction is completed immediately, and an abnormal return code indicates that the task would have been suspended for an indefinite period.

The TPL macro instruction can be used to poll for new connect indications. If a connect indication is available, the request is completed as usual. Otherwise, the request is completed abnormally and the transport user can try again after delaying an appropriate period.

In either case, if a connect indication was already generated, the TPL macro instruction completes normally without suspending the issuing task.

Default: BLOCK (suspend issuing task if necessary).

OPTCD = ASSIGN | USE

Indicates in a TBIND macro instruction whether a protocol address provided by the application program is to be bound to the endpoint, or the transport provider is to assign a local protocol address and return the value to the application program.

OPTCD=ASSIGN if indicated, the transport provider assigns an address and return the value in the storage area designated by the ADLEN and ADBUF operands.

OPTCD=USE if indicated, the ADLEN and ADBUF operands designate a storage area that must contain a valid protocol address.

Default: USE (use protocol address provided).

OPTCD = LOCAL | REMOTE

Indicates in a TADDR macro instruction whether the transport provider should return the local protocol address bound to the indicated endpoint, or the remote protocol address connected to, or associated with, the endpoint.

OPTCD=LOCAL a local address must have already been bound to the endpoint.

OPTCD=REMOTE the endpoint must be connected to (for connection-mode service), or associated with (for connectionless-mode service), a remote protocol address.

If either of these conditions is violated, the TADDR function is completed abnormally.

Default: LOCAL (return local protocol address).

OPTCD = PRIMARY | SECNDRY | STATS

Type of information requested with a TINFO macro instruction.

Valid information types are:

PRIMARY—Designates primary protocol information whose format and meaning is standardized for all transport providers. The application program can use this information to determine the basic characteristics of the transport service and limits of the transport provider.

SECNDRY—Designates secondary protocol information whose format and meaning is specific to the transport service being used. This information includes internal protocol and state variables that govern the operation of the transport protocol. Transport providers are not required to support this option code. Refer to the appropriate appendix at the end of this reference for more information concerning a specific transport provider.

STATS—Designates statistical information recorded by the transport provider whose format and meaning is specific to the transport service being used. Transport providers are not required to support this option code. Refer to the appropriate appendix at the end of this reference for more information concerning a specific transport provider.

Default: PRIMARY (return basic protocol information).

OPTCD = DECLARE | VERIFY | QUERY | DEFAULT

Action that the TOPTION macro instruction should perform. Protocol options that are the subject of these actions are contained or returned in an option list designated by the OPLEN and OPBUF operands. One of these actions may be indicated:

DECLARE—The options specified by the application program are invoked, and the option list is updated with the inferior value of any negotiated options.

VERIFY—The options specified by the application program are verified, and the option list is updated with the inferior value of any negotiated options.

QUERY—The current value of options selected by the application program are returned.

DEFAULT—The default value of options selected by the application program are returned.

The type, number, and format of protocol options supported by a transport provider are protocol specific. Refer to the appropriate appendix at the end of this reference for more information on specific transport providers.

Default: DECLARE (invoke protocol options).

OPTCD = TP | API

Indicates whether the option list identified by the OPBUF and OPLEN operands contains transport interface or transport provider options.

OPTCD=API the option list contains interface options that are processed solely by the API.

OPTCD=TP the option list is passed to the transport provider for processing.

Transport interface and transport provider options can only be manipulated with separate invocations of the TOPTION macro instruction.

Default: TP (transport provider options).

OPTCD = MORE | NOMORE

Indicates for a TSEND macro instruction whether the application program intends to immediately send more data, or pause momentarily until it has more data to send.

OPTCD=MORE the application program expects to immediately issue another TSEND macro instruction at the same endpoint.

OPTCD=NOMORE the application program has no more data to send, but intends to leave the connection established, and may resume sending data later.

The interpretation of this option code by the transport provider is protocol dependent. The intent is that the transport provider uses this information to augment its packetizing algorithm, and deduce when unsent data must be forwarded on the connection. Not all connection-mode transport providers are required to interpret this option code, but all are required to accept it. If the TSEND macro instruction is executed in synchronous mode, OPTCD=MORE is ignored.

No implication is drawn about message boundaries by the indication of OPTCD=NOMORE. If the underlying transport protocol can preserve logical boundaries within the data stream, then such boundaries should be indicated with OPTCD=EOM.

Default: NOMORE (send data immediately).

OPTCD = NORMAL | EXPEDITE	Indicates for a TSEND macro instruction whether the user data should be sent as normal or expedited data. OPTCD=NORMAL the data associated with the request is to be sent as normal data. OPTCD=EXPEDITE the data is to be sent as expedited data. The distinction between normal and expedited data is left to the interpretation of the transport provider. Default: NORMAL (send data as normal data).
OPTCD = EOM | NOTEOM	Indicates whether the data associated with a TSEND or TSENDTO request is a complete message or datagram, or is continued with one or more subsequent macro instructions. OPTCD=EOM the last byte of data corresponds to the end of the message or datagram. OPTCD=NOTEOM the end of the message or datagram does not occur with this request, and is continued with at least one more TSEND or TSENDTO macro instruction. Transport providers operating in connectionless-mode are not required to accept it. If supported, the notion of a message (or TSDU) is synonymous with datagram and the significance of the OPTCD=NOTEOM indication is only a local phenomenon. Default: EOM (end of TSDU or datagram).
OPTCD = DIRECT | INDIR	Format of the user data parameter. OPTCD=DIRECT the DABUF and DALEN operands identify a storage area into which data should be received directly. OPTCD=INDIR the storage area identified by these operands contains an indirect data vector. An indirect data vector consists of a list of address-length pairs, with each element identifying a separate segment of non-contiguous storage. In this case, DABUF is the address of the first element in the list, and DALEN is the total length of the list. **Note:** The length of the vector must be a multiple of eight, and the total amount of data that can be received is the sum of the lengths of each data segment. Default: DIRECT (send directly from data area).

OPTCD = ABORT | CLEAR

Action to be taken by the transport provider when the application program issues a TDISCONN, TREJECT, or TRELACK macro instruction at an endpoint for which a disconnect indication is already pending.

OPTCD=ABOR the request is completed abnormally, and the application program must clear the pending disconnect indication with a TCLEAR macro instruction.

OPTCD=CLEA the TDISCONN, TREJECT, or TRELACK macro instruction is completed normally (or conditionally) if no other errors occur.

Default: CLEAR (clear pending disconnect indication).

OPTCD = DELETE | PASS

Disposition of the endpoint designated in the TPL associated with a TCLOSE macro instruction.

OPTCD=DELETE the endpoint is closed, and any record of the endpoint is deleted from all internal tables and local storage.

OPTCD=PASS the endpoint is not closed, and control of the endpoint is passed to the designated task or address space.

When control is being passed, the TCLOSE request does not complete until a complementary TOPEN (OPTCD=OLD) macro instruction is issued by the acquiring task or address space.

Default: DELETE (delete endpoint).

OPTCD = TUB | ACEE

Format of user ID information referenced by the USER operand.

OPTCD=TUB user ID, group, and password information are provided in a Transport User Block (TUB).

OPTCD=ACEE the user information is contained in an Accessor Environment Element (ACEE) obtained from the local security system.

Default: TUB (user information provided in TUB).

OPTCD = PLAIN | CIPHER

Indicates whether the password contained in the Transport User Block (TUB) designated with the USER operand has been encrypted, or is in its plain text form.

OPTCD=PLAIN the password is in plain text.

OPTCD=CIPHER the password is encrypted.

	The API uses this information when requesting user ID and password verification from the local security system. Default: PLAIN (password in plain text).
OPTCD = NOFULL | FULL	Completion processing for this request based on the amount of data. OPTCD=NOFULL this request is completed as soon as any data arrives. OPTCD=FULL indicates that this request is not to be completed until either a specified timeout occurs, or the requested amount of data arrives to fill up this request. **Note:** Use of OPTCD=FULL requires that you use OPTCD=TIMEOUT. Default: NOFULL.
OPTCD = NOTIMEOUT | TIMEOUT	Completion processing for this request based on time. OPTCD=NOTIMEOUT this request will not be timed. OPTCD=TIMEOUT this request will be completed at the end of a specified time, regardless of the amount of data available. **Note:** Use of OPTCD=TIMEOUT requires that you use OPBUF and OPLEN to specify the timeout option. However, use of OPTCD=TIMEOUT does not require that you use OPTCD=FULL Default: NOTIMEOUT.
OPTCD = MBUF | NOMBUF	DABUF parameter is the address of a TCPaccess MBUF, rather than a data buffer or indirect buffer list. OPTCD=MBUF review the macro using the parameter to determine how the option code will be processed. OPTCD=NOMBUF the DABUF parameter is processed normally. **Note:** This OPTCD is intended only for applications internal to TCPaccess. Default: NOMBUF.

FNCCD = *function_code*

The API function to execute.

Valid values are:

TACCEPT	Accept connection request
TADDR	Get protocol address
TBIND	Bind local protocol address
TCLEAR	Clear disconnect indication
TCLOSE	Close endpoint
TCONFIRM	Receive connect confirmation
TCONNECT	Initiate connection request
TDISCONN	Initiate abortive disconnect
TINFO	Get transport protocol information
TLISTEN	Listen for connect indications
TOPTION	Endpoint option management
TRECV	Receive from connected transport user
TRECVERR	Receive datagram error indication
TRECVFR	Receive a datagram
TREJECT	Reject connection request
TRELACK	Acknowledge orderly release indication
TRELEASE	Initiate or complete orderly release
TRETRACT	Retract a pending TLISTEN request
TSEND	Send to connected transport user
TSENDTO	Send a datagram
TUNBIND	Unbind local protocol address
TUSER	Associate user with endpoint

If a function code is specified, the definition and use of other operands is determined by the designated function. The operands that may be coded, and the rules that apply, are the same as those defined for the API macro instruction that corresponds to the function code. If a function code is not specified, the value already stored in the TPL designates the function to execute.

Default: Not indicated (to be provided later).

ECB = INTERNAL | *event_control_block_addr*

Location of an Event Control Block (ECB) to be posted by the API when the transport service request associated with this TPL completes. The ECB can be any fullword of storage aligned on a fullword boundary.

The ECB and EXIT operands share the same storage location in the TPL, and are therefore mutually exclusive.

If asynchronous mode was specified (OPTCD=ASYNC), the ECB/EXIT field of the TPL (TPLECBXR) is used in this manner:

- If ECB=address is specified, then the API uses the field as the address of an external ECB. The application program is responsible for issuing a TCHECK macro instruction to check and clear this ECB.
- If EXIT=address is specified, then the API uses the field as the address of the TPL exit routine, and schedules the routine as indicated in the following EXIT operand description.
- If ECB=INTERNAL is specified, then the API uses the field as an internal ECB. The application program must issue a TCHECK macro instruction to check and clear this ECB.

If synchronous mode was specified (OPTCD=SYNC), the TPL is flagged to be processed as if ECB=INTERNAL was specified, and the ECB/EXIT field is used as an internal ECB, which is checked and cleared automatically.

The API clears an internal ECB when it starts processing any TPL-based macro instruction, and also when the TPL is checked. However, an external ECB is only cleared when the TPL is checked.

An application program using external ECBs must be sure that the external ECB is cleared before the next TPL-based macro instruction is issued.

For more information about asynchronous processing, see TCPaccess *Assembler API Concepts*, which discusses synchronization and exceptional event handling.

Default: INTERNAL (internal ECB).

EXIT = *tpl_exit_routine_address*

Address of a routine to be scheduled when the request represented by this TPL completes. The EXIT and ECB operands share the same storage location in the TPL, and are therefore mutually exclusive.

The TPL exit routine is scheduled only if asynchronous mode was indicated by OPTCD=ASYNC. If synchronous mode was indicated, the exit routine is not used. If one is specified with this operand, the address

is overwritten with an internal ECB before the request completes. For further information on asynchronous processing, refer to TCPaccess *Assembler API Concepts*.

Default: Not indicated (no TPL exit routine).

MF = (L | M, [*tpl_address*]) List or modify form of the TPL macro instruction.

The second sublist operand, *tpl_address*, is the address of a storage area that contains the Transport Service Parameter List (TPL). If the TPL address is not provided, or the MF operand is not coded, the TPL is generated in line with the macro instruction. If the generate or execute form the TPL macro instruction is desired, the TEXEC macro instruction should be used.

Read List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: Not indicated (nonreentrant, inline list).

Completion Information

If the MF operand is not coded, or MF=L is indicated, the TPL is generated at assembly time, and no executable code is expanded.

Otherwise, the macro instruction expansion contains executable code to generate or modify the TPL. If the macro instruction completes successfully, the general return code in register 15 is set to zero (TROKAY); otherwise, the general return code is set to 12 (TRFATLPL). The function code is returned in register zero, and the TPL address is returned in register one.

Return Codes

The following table lists the symbolic names for the TPL return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code
TROKAY	func. code	n/a
TRFATLPL	func. code	n/a

Usage Information

The TPL macro instruction is used to generate or modify a Transport Service Parameter List (TPL). The TPL may be generated in line with the macro instruction, or remotely in a storage area indicated by the MF operand. The TPL macro instruction is complementary with the TEXEC macro instruction, which is used to execute a TPL.

Any operand that can be specified on other TPL-based macro instructions (except TOPEN) can be specified on the TPL macro instruction. However, this macro instruction does not check for consistency between operands used by different functions.

Example

QLSTN and NEWEP can be specified on the same TPL macro instruction, although QLSTN is used by TBIND, and NEWEP is used by TACCEPT. However, QLSTN and SEQNO cannot be specified together since they both occupy the same location in the TPL. If the latter were attempted, the TPL macro instruction would generate an error at assembly time.

The TPL macro instruction is typically used to generate a TPL that is not specific to a particular function. No function-dependent processing is performed by the TPL macro instruction, even if the FNCCD operand is coded. It is generally advisable to use the list and modify forms of the appropriate TPL-based macro instructions when function-specific parameter lists are required.

The TPL macro instruction can generate a parameter list for any API TPL-based service request, except TOPEN.

Example

The macro instruction TPL QLSTN=5,FNCCD=TBIND,MF=(L,BINDTPL) generates the same parameter list as QLSTN=5,MF=(L,BINDTPL) for the TBIND macro.

The TPL macro instruction is run-to-completion, and a TCHECK macro instruction should never be issued.

TRECV

Receive Normal or Expedited Data on a Connection— The TRECV macro instruction is used to receive normal or expedited data arriving at an endpoint from the connected (or associated) transport user. TRECV is normally used to receive data on an endpoint operating in connection mode, but when supported by the transport provider, datagrams may be received at an endpoint operating in connectionless-mode if the application program has created an association with the transport user.

```
[ symbol ] TRECV [ EP = endpoint_id ]
                 [ ,DALEN = user_data_length ]
                 [ ,DABUF = user_data_address ]
                 [ ,DAALET = user_data_alet ]
                 [ ,OPBUF = protocol_options_address ]
                 [ ,OPLEN = protocol_options_length ]
                 [ ,OPALET = protocol_options_alet ]
                 [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                              [ ,SYNC | ASYNC ]
                              [ ,BLOCK | NOBLOCK ]
                              [ ,DIRECT | INDIR ]
                              [ ,NOPEEK | PEEK ]
                              [ ,NOFULL | FULL ]
                              [ ,NOTIMEOUT | TIME OUT ]
                              [ ,MBUF | NOMBUF ] ) ]
                 [ ,ECB = INTERNAL | event_control_block_addr ]
                 [ ,EXIT = tpl_exit_routine_address ]
                 [ ,MF = ( I | L | G | M | E , [ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TRECV macro instruction executes.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

DALEN = *user_data_length*

Length (in bytes) of a data storage area or an indirect data vector identified by the DABUF operand.

The length is updated when the request is completed to reflect the actual amount of data received. If the value indicated for DALEN is zero, no data is returned to the application program.

Default: Zero (return no user data).

DABUF = *user_data_address*

Address of a storage area for receiving data that has arrived at the endpoint.

If the data mode is direct, then the value specified is the address of a contiguous storage area for receiving the data.

If the data mode is not direct, then the value specified must be the address of an indirect data vector, and each element of the vector must have been initialized to point to an individual segment of non-contiguous storage.

All available data is moved into the storage area, and the length of the storage area is updated to indicate the amount of data received. If more data is available than fits in the storage area provided, the storage area is filled, and the remaining data is held by the transport provider until another TRECV macro instruction is issued.

All user data received is application-dependent, and is not interpreted by the API or the transport provider. The maximum amount of data that can be received with a single TRECV macro instruction is provider-dependent, and can be determined with the TINFO macro instruction.

The storage area can be aligned on any boundary convenient for the application program.

Default: Zero (no user data storage area).

DAALET = *user_data_alet*

Access List Entry Token (ALET) used in access register (AR) mode when referencing the storage specified by the DABUF parameter.

The DAALET value must be an ALET contained in the Dispatchable Unit Access List (DUAL) of the caller.

The DAALET parameter may be used only if OPTCD=EXTEND is also specified.

Default: Zero (the storage is contained in the address space of the caller).

OPLEN = *protocol_options_length*

Length (in bytes) of the protocol option list identified by the OPBUF operand.

A value of zero indicates there is no protocol option list.

Default: Zero (no protocol option list).

OPBUF = *protocol_options_address*

Address of a storage area containing a protocol option list. The area must contain a list of variable-length protocol options, with each option identified by its length and name. Each entry in the list must also contain room for an options value.

The type, number, and format of protocol options are provider-dependent and the maximum size of the option list can be determined by issuing a TINFO macro instruction.

The storage area can be aligned on any boundary convenient to the application program.

Default: Zero (no protocol option list).

Note: The only option currently supported on the TRECV statement is TPOPRTIM.

OPALET = *protocol_options_alet*

Access List Entry Token (ALET) used in access register (AR) mode when referencing the storage specified by the OPBUF parameter.

The OPALET value must be an ALET contained in the Dispatchable Unit Access List (DUAL) of the caller.

The OPALET parameter may be used only if OPTCD=EXTEND is also specified.

Default: Zero (the storage is contained in the address space of the caller).

OPTCD = SHORT | LONG | EXTEND

Format attribute of the parameter list associated with this request.

OPTCD=SHORT a different control block identifier is used to indicate that a subset of the TPL was generated.

OPTCD=LONG a standard, full-length TPL is generated.

OPTCD=EXEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters.

Default: SHORT if MF=I or MF operand omitted, LONG otherwise.

OPTCD = SYNC | ASYNC — Synchronization mode to use when executing the TRECV macro instruction.

OPTCD=SYNC the request executes in synchronous mode and control is not returned to the application program until the requested macro instruction is complete. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API.

OPTCD=ASYNC the request is executed in asynchronous mode and control is returned immediately after scheduling the TRECV request. The application program is responsible for issuing the TCHECK macro instruction.

Default: SYNC (synchronous mode).

OPTCD = BLOCK | NOBLOCK — Indicates whether the issuing task can be suspended if the TRECV macro instruction cannot be completed immediately.

OPTCD=BLOCK(SYNC is also set, and no data is available to be received) the issuing task is suspended until more data is received at the endpoint.

OPTCD=NOBLOCK the TRECV macro instruction is completed immediately, and an abnormal return code indicates that the task would have been suspended for an indefinite period.

In either case, if data is available to be received, the TRECV macro instruction completes normally without suspending the issuing task. When OPTCD=NOBLOCK is indicated, the TRECV macro instruction can be used to poll for available data. If user data was already received, and is available, the request is completed as usual. Otherwise, the request is completed abnormally, and the transport user can try again after delaying an appropriate period.

Default: BLOCK (suspend issuing task if necessary).

OPTCD = DIRECT | INDIR

Format of the user data parameter.

OPTCD=DIRECT the DABUF and DALEN operands identify a storage area into which data should be received directly.

OPTCD=INDIR the storage area identified by these operands contains an indirect data vector.

An indirect data vector consists of a list of address-length pairs, with each element identifying a separate segment of non-contiguous storage. In this case, DABUF is the address of the first element in the list, and DALEN is the total length of the list.

The length of the vector must be a multiple of eight, and the total amount of data that can be received is the sum of the lengths of each data segment.

Default: DIRECT (receive directly into data area).

OPTCD = NOPEEK | PEEK

Indicates whether or not received data will be consumed or browsed.

OPTCD = NOPEEK the TRECV causes the received data to be consumed. That is, the data received into the buffer will be counted against the total number of bytes being received; it cannot be re-received.

OPTCD = PEEK the data received into the buffer is effectively browsed; it is not consumed. The overall length of data to be received is not decremented. The next non-PEEK will consume the data.

Default: NOPEEK (consume data).

OPTCD = NOFULL | FULL

Completion processing for this request based on the amount of data.

OPTCD=NOFULL this request is completed as soon as any data arrives.

OPTCD=FULL indicates that this request is not to be completed until either a specified timeout occurs, or the requested amount of data arrives to fill up this request.

Note: Use of OPTCD=FULL requires that you use OPTCD=TIMEOUT.

Default: NOFULL.

OPTCD = NOTIMEOUT | TIMEOUT	Completion processing for this request based on time. OPTCD=NOTIMEOUT this request will not be timed. OPTCD=TIMEOUT this request will be completed at the end of a specified time, regardless of the amount of data available. **Note:** Use of OPTCD=TIMEOUT requires that you use OPBUF and OPLEN to specify the timeout option. However, use of OPTCD=TIMEOUT does not require that you use OPTCD=FULL. Default: NOTIMEOUT.
OPTCD = MBUF | NOMBUF	How the DABUF parameter is handled. OPTCD = MBUF the DABUF parameter is ignored and the MBUF address is placed in TPLDABUF upon the completion of the request. OPTCD = NOMBUF the DABUF parameter is processed normally. **Note:** This OPTCD is intended only for applications internal to TCPaccess. Default: NOMBUF.
ECB = INTERNAL | *event_control_block_addr*	Location of an Event Control Block (ECB) to be posted by the API when the TRECV macro instruction associated with this TPL completes. The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB. The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB. This operand is mutually exclusive with the following EXIT operand. Default: INTERNAL (internal ECB).

EXIT = *tpl_exit_routine_address*	Address of an exit routine to be scheduled when the TRECV macro instruction associated with this TPL completes. The TPL exit routine is scheduled only if asynchronous mode was specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB. This operand is mutually exclusive with the previous ECB operand. Default: Not indicated (no TPL exit routine).
MF = (I | L | G | M | E , [*tpl_address*])	Standard, list, generate, modify, or execute form of the TRECV macro instruction. The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used. Refer to List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters. Default: MF=I (standard, nonreentrant form).

Completion Information

The TRECV macro instruction completes normally when data is available at the endpoint, and has been moved into the storage area provided by the application program. The length of the storage area is updated to indicate the amount of data received.

The following is a list of the options set in the TPL OPTCD field on return to the application program:

NOTEOM	There is more data, and the current transport service data unit must be received with multiple TRECV macro instructions. If the transport provider does not support this concept of a TSDU, NOTEOM is not indicated.
EOM (NOTEOM not set)	The last byte of data received corresponds to the end of the TSDU.
MORE	More data is buffered for the endpoint, regardless of whether the transport provider supports the concept of a TSDU. The data may be a continuation of the current TSDU, or the beginning of the next TSDU.
NOMORE (MORE not set)	No data is buffered.
EXPEDITE	The data is expedited data.
NORMAL (EXPEDITE not set)	The data is normal data.

NOTEOM and MORE apply to expedited data in the same way they apply to normal data. If NOTEOM and EXPEDITE are indicated together, the data is the beginning or continuation of an expedited transport service data unit that must be received with multiple TRECV macro instructions. If EOM and EXPEDITE are indicated, the data comprises the end of an ETSDU.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY). The conditional completion code in register zero is always zero (TCOKAY), and the TPL return code field is set accordingly. No other information is returned.

If the TRECV macro instruction completes abnormally, no data is moved into the storage area, and the OPTCD indicators are not set. The state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TRECV return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY	TCTIME	
TRFAILED	TAINTEG	TEPROTO	TEDISCON	TERELESE
	TAENVIRO	TESYSERR TEUNSUPF	TESUBSYS TESTOP	TEDRAIN TETERM
	TAFORMAT	TEBDFNCD TEBDDATA	TEBDOPCD TEBDEXIT	TEBDECB TEBDOPTN
	TAPROCED	TEAMODE TEBUFOVR	TESTATE	TEREQOVR
	TATPLERR	TEACTIVE		
TRFATLFC	func. code	The function code loaded into register zero is invalid.		
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.		
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.		
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.		

Usage Information

The TRECV macro instruction is used to receive normal or expedited data that has arrived at an endpoint. The data may be part of a byte stream or message stream sent over a connection, or part (or all) of a datagram received via an association with the remote transport user.

If the transport service type is a connection-mode byte stream, data buffered at the endpoint is moved into the storage area provided by the application program. If the total amount of buffered data is less than the size of the storage area, all data is moved and the length of the storage area is updated to reflect the actual amount of data moved. NOMORE is also set in the TPL to indicate no more data is buffered at the endpoint. Otherwise, the storage area is filled to capacity, and MORE is set to indicate that more data is ready to be received.

If the transport service type is a connection-mode message stream, processing is similar. However, the transport provider must preserve the logical boundary of TSDUs, and indicate where the boundaries occur. If the data moved into the storage area comprises the end of the TSDU, EOM is set in the TPL to indicate that the TSDU was completely received. Otherwise, NOTEOM is set to indicate another TRECV macro instruction should be issued to receive the end of the TSDU.

If the buffered data contains the end of one TSDU and the beginning of the next TSDU, only data up to the end of the current TSDU is moved into the storage area. In other words, data from two TSDUs is never moved into the storage area at one time. Another TRECV macro instruction must be executed to receive the next TSDU. MORE is set to indicate another TSDU is available to be received.

If the transport service type is connectionless-mode using associations, the datagram is handled like a TSDU. If the entire datagram does not fit within the storage area, NOTEOM is set in the TPL indicating that another TRECV macro instruction must be issued to receive the continuation of the datagram. If the data moved contains the end of the datagram, and another datagram is available at the endpoint, MORE is set to indicate a new datagram exists.

If the transport provider supports expedited data, the same macro instruction is used to receive it. TOEXPDTE is set in the TPL to distinguish normal data from expedited data. In a manner similar to the handling of TSDUs, normal and expedited data is never mixed. Datagrams are always classified as normal data.

The application program does not ask to receive expedited data; rather, it requests to receive data, and the API indicates with the TOEXPDTE flag whether the data received was expedited or normal. The distinction between normal and expedited data is provider-dependent. Some transport providers can process expedited data out-of-band, and if normal and expedited data are available at the same time, expedited data is delivered ahead of normal data. However, an expedited TSDU can never interrupt a normal TSDU. That is, a normal TSDU must be received in its entirety before the expedited TSDU can be received. Transport providers that do not support out-of-band expedited data must deliver the data in sequence (for example, TCP urgent data). In this case, the application program should use the indication to expedite processing of the data stream.

TCP Provider Session Options

These options are valid only for TCP provider sessions. Each of these values is four bytes long, so the option length must be eight. Refer to TOPTION for format information.

If the option is TPOPRTIM, the time, in seconds, to wait for data to arrive to satisfy this request. This option is valid only when specified on a TRECV TPL.

Note: TPOPRTIM *must* be specified with OPTCD=TIMEOUT. It is possible for a TRECV to complete with zero bytes of data, or with less than a full request if used with OPTCD=FULL. The return values of TAOKAY and TCTIME indicate this situation.

Return Indicators

The OPTCD field of the TPL is used by the API data transfer routines to return indicators that can be tested by the application program. The indicators returned by the TRECV macro instruction correspond to the indicators set by the TSEND macro instruction. These indicators are located in the function-specific option code field mapped by the TPL DSECT as TPLOPCD2.

These bits are used by the TRECV macro instruction:

- If set, the TONOTEOM bit corresponds to OPTCD=NOTEOM, and indicates that another TRECV macro instruction must be issued to receive the end of the TSDU.

  If not set, the TONOTEOM bit corresponds to OPTCD=EOM, and indicates the last byte of data moved into the storage area corresponds to the end of the TSDU.

- If set, the TOMORE bit corresponds to OPTCD=MORE and indicates that more data is buffered at the endpoint. The data is not necessarily part of the current TSDU.

If not set, the TOMORE bit corresponds to OPTCD=NOMORE, and indicates all of the data buffered at the endpoint has been moved into the application program's storage area.

- If set, the TOEXPDTE bit corresponds to OPTCD=EXPEDITE, and indicates that the data moved into the storage area is expedited data.

  If not set, the TOEXPDTE bit corresponds to OPTCD=NORMAL, and indicates that the data moved into the storage area is normal data. If the transport provider supports the concept of an expedited transport service data unit, TONOTEOM is used to mark the continuation of the ETSDU.

The storage area provided by the application program can be a simple, contiguous segment of storage, or a set of non-contiguous segments indirectly addressed via a data vector.

- If the option is OPTCD=DIRECT, user data is transferred into the storage area identified by the DABUF and DALEN operands. The length of the storage area is updated to reflect the actual amount of data transferred.
- If the option is OPTCD=INDIR, DABUF and DALEN identify a storage area initialized with the addresses and lengths of non-contiguous storage segments into which the user data is to be transferred. The length of the indirect data vector is updated to reflect the actual amount of data transferred.

Each entry in an indirect data vector consists of a fullword address followed by a fullword length. If the length is zero, the entry is ignored. If the length is nonzero, the address must reference a valid storage area, and can be aligned on any boundary convenient for the application program. The length of the vector is used to determine the number of entries in the list. The sum of the lengths must not exceed the maximum interface data unit defined for the endpoint.

Unlike most other macro instructions, multiple TRECV macro instructions can be issued without waiting for the first to complete. However, each macro instruction requires its own TPL. The maximum number that can be issued before one must complete is an API variable that can be modified by the TOPTION macro instruction. The default value is set when the API is installed. TRECV macro instructions are completed in the order in which they are issued.

Data received with the TRECV macro instruction is buffered in the API address space before it is moved into the storage area provided by the application program. The total amount of receive buffering allocated for an endpoint is also an API option. For TLI-mode sockets, zero (0) is returned in the residual count field. For socket mode, the residual count returned when the TRECV macro instruction completes is set to the total number of bytes placed into the data buffers.

The data received is application-dependent and is not interpreted by the API or the transport provider. You can determine the maximum amount that can be received with a single TRECV request by issuing a TINFO macro instruction.

TRECVERR

Receive Datagram Error Indication— The TRECVERR macro instruction is used to receive an error indication associated with a datagram previously sent on an endpoint operating in connectionless-mode. A protocol-specific datagram error code is returned to the application program, as well as the remote protocol address.

```
[ symbol ] TRECVERR [ EP = endpoint_id ]
                    [ ,ADLEN = protocol_address_length ]
                    [ ,ADBUF = protocol_address_address ]
                    [ ,ADALET = protocol_address_alet ]
                    [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                                 [ ,SYNC | ASYNC ]
                                 [ ,TRUNC | NOTRUNC ] ) ]
                    [ ,ECB = INTERNAL | event_control_block_addr ]
                    [ ,EXIT = tpl_exit_routine_address ]
                    [ ,MF = ( I | L | G | M | E , [ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TRECVERR macro instruction is executed.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

ADLEN = *protocol_address_length*

Length (in bytes) of the protocol address storage area identified by the ADBUF operand.

The length is updated when the request is completed to reflect the actual length of the protocol address returned. If the length is zero, the protocol address of the remote transport user is not returned to the application program.

Default: Zero (return no protocol address).

ADBUF = *protocol_address_address*	Address of a storage area for returning the protocol address of the remote transport user. The storage area should be large enough to contain the entire address. The format of the protocol address is provider-dependent, and its maximum size can be determined by issuing a TINFO macro instruction. The storage area can be aligned on any boundary. Default: Zero (no protocol address storage area).
ADALET = *protocol_address_alet*	Access List Entry Token (ALET) used in access register (AR) mode when referencing the storage specified by the ADBUF parameter. The ADALET value must be an ALET contained in the Dispatchable Unit Access List (DUAL) of the caller. The ADALET parameter can be used only if OPTCD=EXTEND is also specified. Default: Zero (the storage is contained in the address space of the caller).
OPTCD = SHORT | LONG | EXTEND	Format attribute of the parameter list associated with this request. OPTCD=SHORT a different control block identifier is used to indicate that a subset of the TPL was generated. OPTCD=LONG a standard, full-length TPL is generated. OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters. Default: SHORT if MF=I or MF operand omitted, LONG otherwise.
OPTCD = SYNC | ASYNC	Synchronization mode to use when executing the TRECVERR macro instruction. OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction is complete. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API. OPTCD=ASYNC the request executes in asynchronous mode, and returns control immediately after scheduling the TRECVERR request. The application program is responsible for issuing the TCHECK macro instruction. Default: SYNC (synchronous mode).

OPTCD = TRUNC | NOTRUNC	Indicates whether the protocol address or options returned to the application program by the transport provider should be truncated if they do not fit within the storage area provided. OPTCD=TRUNC the excess is truncated, and the TRECVERR macro instruction is completed conditionally as long as no other errors occur. OPTCD=NOTRUNC nothing is placed in the storage area, and the TRECVERR macro instruction is completed abnormally. Default: NOTRUNC (no truncation)
ECB = INTERNAL | *event_control_block_address*	Location of an Event Control Block (ECB) to be posted by the API when the TRECVERR macro instruction associated with this TPL completes. The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB. The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB. This operand is mutually exclusive with the following EXIT operand. Default: INTERNAL (internal ECB).
EXIT = *tpl_exit_routine_address*	Address of an exit routine to schedule when the TRECVERR macro instruction associated with this TPL completes. The TPL exit routine is scheduled only if asynchronous mode was specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB. This operand is mutually exclusive with the previous ECB operand. Default: Not indicated (no TPL exit routine).

MF = (I | L | G | M | E, [*tpl_address*])

Standard, list, generate, modify, or execute form of the TRECVERR macro instruction.

The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used.

See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: MF=I (standard, nonreentrant form).

Completion Information

The TRECVERR macro instruction completes normally (or conditionally) when the information associated with a datagram error indication has been moved into the storage areas provided by the application program. The protocol address of the remote transport user is returned, and the storage length is updated to reflect the amount of information returned. A protocol-dependent error code is also returned in the DGERR field of the TPL associated with this request.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY), and a conditional completion code is returned in register zero. TCTRUNC is set if the protocol address returned to the application program was truncated to fit in the storage area provided. The TPL return code field is set accordingly. No other information is returned.

If the TRECVERR macro instruction completes abnormally, no information is returned to the application program, and the datagram error indication (if any) remains pending. The state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TRECVERR return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY	TCTRUNC	
TRFAILED	TAINTEG	TEOVRFLO		
	TAENVIRO	TESYSERR TESTOP TEUNSUPF	TESUBSYS TETERM	TEDRAIN TEUNSUPO
	TAFORMAT	TEBDFNCD TEBDEXIT	TEBDOPCD TEBADDR	TEBDECB
	TAPROCED	TEAMODE	TESTATE	TENOERR
	TATPLERR	TEACTIVE		
TRFATLFC	func. code	The function code loaded into register zero is invalid.		
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.		
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.		
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.		

Usage Information

The TRECVERR macro instruction is used to receive information associated with an error that occurred with a previously sent datagram. The transport provider returns the remote protocol address and options associated with the datagram, and a protocol-dependent error code that identifies the specific error.

The error occurred after being sent with a TSENDTO macro instruction that completed normally. When the error is detected, the transport provider generates a datagram error indication. The TRECVERR macro instruction clears the pending indication and receives the information associated with the error.

TRECVFR

Receive a Datagram — The TRECVFR macro instruction is used to receive datagrams arriving at an endpoint operating in connectionless-mode. The user data and the remote protocol address of the sender are returned to the application program.

```
[ symbol ] TRECVFR [ EP = endpoint_id ]
                   [ ,ADLEN = protocol_address_length ]
                   [ ,ADBUF = protocol_address_address ]
                   [ ,ADALET = protocol_address_alet ]
                   [ ,DALEN = user_data_length ]
                   [ ,DABUF = user_data_address ]
                   [ ,DAALET = user_data_alet ]
                   [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                                [ ,SYNC | ASYNC ]
                                [ ,TRUNC | NOTRUNC ]
                                [ ,BLOCK | NOBLOCK ]
                                [ ,DIRECT | INDIR ]
                                [ ,MBUF | NOMBUF ] ) ]
                   [ ,ECB = INTERNAL | event_control_block_addr ]
                   [ ,EXIT = tpl_exit_routine_address ]
                   [ ,MF = ( I | L | G | M | E , [ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TRECVFR macro instruction executes.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

ADLEN = *protocol_address_length*

Length (in bytes) of the protocol address storage area identified by the ADBUF operand.

When the request completes, the length is updated to reflect the actual length of the protocol address returned. If the length is zero, the protocol address of the sending transport user is not returned to the application program.

Default: Zero (return no protocol address).

ADBUF = *protocol_address_address*

Address of a storage area for returning the protocol address of the sending transport user.

The storage area should be large enough to contain the entire address. The format of the protocol address is provider-dependent, and its maximum size can be determined by issuing a TINFO macro instruction. The storage area can be aligned on any boundary.

Default: Zero (no protocol address storage area).

ADALET = *protocol_address_alet*

Access List Entry Token (ALET) used in access register (AR) mode when referencing the storage specified by the ADBUF parameter.

The ADALET value must be an ALET contained in the Dispatchable Unit Access List (DUAL) of the caller. The ADALET parameter can be used only if OPTCD=EXTEND is also specified.

Default: Zero (the storage is contained in the address space of the caller).

DALEN = *user_data_length*

Length (in bytes) of a data storage area or an indirect data vector identified by the DABUF operand.

When the request completes, the length is updated to reflect the actual amount of data received.

If the value indicated for DALEN is zero, no data is returned to the application program.

Default: Zero (return no user data).

DABUF = *user_data_address*

Address of a storage area for receiving data that has arrived at the endpoint.

If the data mode is direct, the value specified is the address of a contiguous storage area for receiving the data.

If the data mode is not direct, the value specified must be the address of an indirect data vector, and each element of the vector must have been initialized to point to an individual segment of non-contiguous storage.

If more data is available than fits in the storage area provided, the storage area is filled, and the remaining data is held by the transport provider until another TRECVFR macro instruction is issued. Otherwise, all available data is moved into the storage area, and the length of the storage area is updated to indicate the amount of data received.

All user data received is application-dependent, and is not interpreted by the API or the transport provider. The maximum amount of data that can be received with a single TRECVFR macro instruction is provider-dependent, and can be determined with the TINFO macro instruction. The storage area can be aligned on any boundary convenient for the application program.

Default: Zero (no user data storage area).

DAALET = *user_data_alet*

Access List Entry Token (ALET) used in access register (AR) mode when referencing the storage specified by the DABUF parameter.

The DAALET value must be an ALET contained in the Dispatchable Unit Access List (DUAL) of the caller. The DAALET parameter can be used only if OPTCD=EXTEND is also specified.

Default: Zero (the storage is contained in the address space of the caller).

OPTCD = SHORT | LONG | EXTEND	Format attribute of the parameter list associated with this request. OPTCD=SHORT a different control block identifier used to indicate that a subset of the TPL was generated. OPTCD=LONG a standard, full-length TPL is generated. OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters. Default: SHORT if MF=I or MF operand omitted, LONG otherwise.
OPTCD = SYNC | ASYNC	Synchronization mode to use when executing the TRECVFR macro instruction. OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction is complete. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API. OPTCD=ASYNC the request executes in asynchronous mode, and returns control immediately after scheduling the TRECVFR request. The application program is responsible for issuing the TCHECK macro instruction. Default: SYNC (synchronous mode).
OPTCD = TRUNC | NOTRUNC	Indicates whether the protocol address returned to the application program by the transport provider should be truncated if it does not fit within the storage area provided. OPTCD=TRUNC the excess is truncated, and the TRECVFR macro instruction completes conditionally as long as no other errors occur. OPTCD=NOTRUNC nothing is placed in the storage area, and the TRECVFR macro instruction completes abnormally. Default: NOTRUNC (no truncation).

OPTCD = BLOCK | NOBLOCK	Indicates whether the issuing task can be suspended if the TRECVFR macro instruction cannot be completed immediately. OPTCD=BLOCK (and no data is available to be received) the issuing task is suspended until more data is received at the endpoint. OPTCD=NOBLOCK the TRECVFR macro instruction completes immediately, and an abnormal return code indicates that the task would have been suspended for an indefinite period. When OPTCD=NOBLOCK is specified, the TRECVFR macro instruction can be used to poll for available datagrams. If a datagram was received and is available, the request is completed as usual. Otherwise, the request is completed abnormally, and the transport user can try again after delaying an appropriate period. In either case, if a datagram is available to be received, the TRECVFR macro instruction completes normally without suspending the issuing task. Default: BLOCK (suspend issuing task if necessary).
OPTCD = DIRECT | INDIR	Format of the user data parameter. OPTCD=DIRECT the DABUF and DALEN operands identify a storage area into which data should be received directly. OPTCD=INDIR the storage area identified by these operands contains an indirect data vector. An indirect data vector consists of a list of address-length pairs, with each element identifying a separate segment of non-contiguous storage. In this case, DABUF is the address of the first element in the list, and DALEN is the total length of the list. The length of the vector must be a multiple of eight, and the total amount of data that can be received is the sum of the lengths of each data segment. Default: DIRECT (receive directly into data area).
OPTCD = MBUF | NOMBUF	How to handle the DABUF parameter. OPTCD=MBUF the DABUF parameter is ignored and the MBUF address is placed in TPLDABUF upon the completion of the request. OPTCD=NOMBUF the DABUF parameter is processed normally. **Note:** This OPTCD is intended only for applications internal to TCPaccess. Default: NOMBUF.

ECB = INTERNAL | *event_control_block_addr*

Location of an Event Control Block (ECB) to be posted by the API when the TRECVFR macro instruction associated with this TPL completes.

The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB.

The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB.

This operand is mutually exclusive with the following EXIT operand.

Default: INTERNAL (internal ECB).

EXIT = *tpl_exit_routine_address*

Address of an exit routine to schedule when the TRECVFR macro instruction associated with this TPL completes.

The TPL exit routine is scheduled only if asynchronous mode was specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB.

This operand is mutually exclusive with the previous ECB operand.

Default: Not indicated (no TPL exit routine).

MF = (I | L | G | M | E, [*tpl_address*])

Standard, list, generate, modify, or execute form of the TRECVFR macro instruction.

The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used.

See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: MF=I (standard, nonreentrant form).

Completion Information

The TRECVFR macro instruction completes normally (or conditionally) when data is available at the endpoint, and has been moved into the storage area provided by the application program. The length of the storage area is updated to reflect the amount of data received with this request. A residual count is also returned which is set to the amount of internal buffer space for which no receive is pending.

If the data returned to the application program is the beginning of a new datagram, the protocol address of the sender is returned if a storage area was provided. The corresponding lengths are updated to reflect the actual amount of data returned. If the data is the continuation of an old datagram, the lengths of these storage areas are set to zero.

On return to the application program, TONOTEOM is indicated in the field if there is more data, and the current datagram must be received with multiple TRECVFR macro instructions. If end of message is indicated in the OPTCD field (that is, TONOTEOM not set), the last byte of the data received corresponds to the end of the datagram.

TOMORE is indicated in the OPTCD field if more data is buffered for the endpoint. The data may be a continuation of the current datagram, or the beginning of the next datagram. If no data is buffered, TOMORE is not set in the TPLOPTCD field.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY), and a conditional completion code is returned in register zero. TCTRUNC is set if the protocol address returned to the application program was truncated to fit in the storage area provided. The TPL return code field is set accordingly. No other information is returned.

If the TRECVFR macro instruction completes abnormally, no data is moved into the storage area, and the OPTCD indicators described above are not set. The state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TRECVFR return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY	TCTRUNC	
TRFAILED	TAINTEG	TEPROTO	TEOVRFLO	
	TAENVIRO	TESYSERR TESTOP TEUNSUPF	TESUBSYS TETERM	TEDRAIN TEUNSUPO
	TAFORMAT	TEBDFNCD TEBDEXIT	TEBDOPCD TEBDADDR	TEBDECB TEBDDATA
	TAPROCED	TEAMODE TEBUFOVR	TESTATE	TEREQOVR
	TATPLERR	TEACTIVE		
TRFATLFC	func. code	The function code loaded into register zero is invalid.		
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.		
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.		
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.		

Usage Information

The TRECVFR macro instruction is used to receive a datagram that has arrived at an endpoint operating in connectionless mode. The protocol address of the sending transport user, protocol options associated with the datagram, and user data contained within the datagram itself are returned to the application program. The endpoint must be in the disabled (TSDSABLD) state when the TRECVFR macro instruction is issued.

A datagram can span several TRECVFR macro instructions if the receiving storage area is not large enough to hold the entire amount of data. If the storage area is filled to capacity, but more data remains for the current datagram, NOTEOM is set in the TPL to indicate another TRECVFR macro instruction is required to receive the continuation of the datagram. This indication continues to be set until the end of the datagram is received, at which time EOM is indicated. The protocol address and any protocol options associated with the datagram are returned with the first segment of the datagram.

If another datagram is available to be received, MORE is indicated at the completion of the TRECVFR macro instruction. Otherwise, NOMORE is indicated. Data from two different datagrams is never combined, even if both could fit within the storage area provided.

The TPLOPTCD field of the TPL is used by the API data transfer routines to return indicators that can be tested by the application program. The indicators returned by the TRECVFR macro instruction correspond to the indicators set by the TSENDTO macro instruction. These indicators are located in the function-specific option code field mapped by the TPL DSECT as TPLOPCD2.

TONOTEOM If set, corresponds to OPTCD=NOTEOM, and indicates that another TRECVFR macro instruction must be issued to receive the end of the datagram.

If not set, corresponds to OPTCD=EOM, and indicates the last byte of data moved into the storage area corresponds to the end of the datagram.

TOMORE If set, corresponds to OPTCD=MORE, and indicates that more data is buffered at the endpoint. The data is not necessarily part of the current datagram.

If not set, corresponds to OPTCD=NOMORE, and indicates all of the data buffered at the endpoint was moved into the application program's storage area.

The storage area provided by the application program can be a simple, contiguous segment of storage, or a set of non-contiguous segments indirectly addressed via a data vector.

If the option is:

- OPTCD=DIRECT—The datagram is transferred into the storage area identified by the DABUF and DALEN operands. The length of the storage area is updated to reflect the actual amount of data transferred.
- OPTCD=INDIR—The DABUF and DALEN operands identify a storage area initialized with the addresses and lengths of non-contiguous storage segments into which the datagram is to be transferred. The length of the indirect data vector is updated to reflect the actual amount of data transferred.

Each entry in an indirect data vector consists of a fullword address followed by a fullword length. If the length is zero, the entry is ignored. If the length is nonzero, the address must reference a valid storage area, and may be aligned on any boundary convenient for the application program. The length of the vector is used to determine the number of entries in the list. The sum of the lengths must not exceed the maximum interface data unit defined for the endpoint.

Unlike most other macro instructions, multiple TRECVFR macro instructions can be issued without waiting for the first to complete. However, each macro instruction requires its own TPL. The maximum number that can be issued before one must complete is an API variable that can be modified by the TOPTION macro instruction. The default value is set when the API is configured. TRECVFR macro instructions are completed in the order in which they are issued.

Data received with the TRECVFR macro instruction is buffered in the API address space before it is moved into the storage area provided by the application program. The total amount of receive buffering allocated for an endpoint is also an API option.

The data received is application-dependent, and is not interpreted by the API or the transport provider. The maximum amount that can be received with a single TRECVFR request can be determined by issuing a TINFO macro instruction.

TREJECT

Reject a Connection Request—When a connect indication has been received at an endpoint with a TLISTEN macro instruction, the TREJECT macro instruction is used to reject the connection request.

```
[ symbol ] TREJECT [ EP = endpoint_id ]
                   [ ,SEQNO = sequence_number ]
                   [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                                [ ,SYNC | ASYNC ]
                                [ ,ABORT | CLEAR ] ) ]
                   [ ,ECB = INTERNAL | event_control_block_addr ]
                   [ ,EXIT = tpl_exit_routine_address ]
                   [ ,MF = ( I | L | G | M | E,[ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TREJECT macro instruction executes

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

SEQNO = *sequence_number*

Which connect indication is rejected.

The value specified must have been returned by a TLISTEN macro instruction. The transport provider uses this value to identify a connect indication pending for this endpoint, which has not yet been accepted or rejected.

Default: Zero (most likely an invalid sequence number).

OPTCD = SHORT | LONG | EXTEND

Format attribute of the parameter list associated with this request.

OPTCD=SHORT a different control block identifier used to indicate that a subset of the TPL was generated.

OPTCD=LONG a standard, full-length TPL is generated.

OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters.

Default: SHORT if MF=I or MF operand omitted, LONG otherwise.

OPTCD = SYNC | ASYNC

Synchronization mode to use when executing the TREJECT macro instruction.

OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction completes. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API.

OPTCD=ASYNC the request executes in asynchronous mode, and control is returned immediately after scheduling the TREJECT request. The application program is responsible for issuing the TCHECK macro instruction.

Default: SYNC (synchronous mode).

OPTCD = ABORT | CLEAR

Action that the transport provider should take when the application program issues a TREJECT macro instruction at an endpoint for which a disconnect indication is already pending.

OPTCD=ABORT the request completes abnormally, and the application program must clear the pending disconnect indication with a TCLEAR macro instruction.

OPTCD=CLEAR the TREJECT request clears the disconnect indication, and the macro instruction completes normally (or conditionally) if no other errors occur.

Default: CLEAR (clear pending disconnect indication).

ECB = INTERNAL | *event_control_block_addr*

Location of an Event Control Block (ECB) to be posted by the API when the TREJECT macro instruction associated with this TPL completes.

The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB.

The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB.

This operand is mutually exclusive with the following EXIT operand.

Default: INTERNAL (internal ECB).

EXIT = *tpl_exit_routine_address*

Address of an exit routine to schedule when the TREJECT macro instruction associated with this TPL completes.

The TPL exit routine is scheduled only if asynchronous mode has been specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB.

This operand is mutually exclusive with the previous ECB operand.

Default: Not indicated (no TPL exit routine).

MF = (I | L | G | M | E , [*tpl_address*])

Atandard, list, generate, modify, or execute form of the TREJECT macro instruction.

The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used.

See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: MF=I (standard, nonreentrant form).

Completion Information

The TREJECT macro instruction completes normally when the disconnect protocol data unit has been scheduled by the transport provider for transmission to the calling transport user, and the pending connect indication is removed from the queue. The state of the endpoint is changed from connect-indication-pending (TSINCONN) to enabled (TSENABLD) if no other indications are pending. Otherwise, the state of the endpoint is unchanged.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY). The conditional completion code in register zero is always zero (TCOKAY), and the TPL return code field is set accordingly. No other information is returned.

If the TREJECT macro instruction completes abnormally, no disconnect protocol data unit is scheduled for transmission to the calling transport user, and the connect indication remains pending. The state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TREJECT return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY		
		TAINTEG	TEPROTO	TEDISCON
	TAENVIRO	TESYSERR TESTOP TEUNSUPF	TESUBSYS TETERM	TEDRAIN TEUNSUPO
	TAFORMAT	TEBDFNCD TEBDEXIT	TEBDOPCD TEBDDATA	TEBDECB TEBDSQNO
	TAPROCED	TEAMODE	TESTATE	TEINCMPL
	TATPLERR	TEACTIVE		
TRFATLFC	func. code	The function code loaded into register zero is invalid.		
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.		
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.		
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.		

Usage Information

The TREJECT macro instruction is used to refuse a connection request received from a remote transport user.

The TREJECT macro instruction is issued in place of a TACCEPT macro instruction to reject a pending connect indication, and to remove it from the queue. The sequence number returned with a TLISTEN macro instruction must be provided to designate which connect indication is being rejected. The endpoint must be in the connect-indication-pending (TSINCONN) state, and remains in this state if more connect indications are pending. Otherwise, the endpoint is returned to the enabled (TSENABLD) state, and remains enabled to receive more connect indications.

The TREJECT macro instruction is usually executed at endpoints operating in connection mode. However, if a TLISTEN macro instruction has been executed at an endpoint operating in connectionless mode, the TREJECT macro instruction may be issued to reject a simulated connect indication. The connect indication was generated as the result of receiving a datagram at the endpoint, and the TREJECT macro instruction causes the indication to be removed, and the associated datagram to be discarded. No association is established, and another TLISTEN macro instruction should be issued to receive the next indication when a datagram arrives.

TRELACK

Acknowledge Orderly Release Indication— The TRELACK macro instruction is used to acknowledge an indication of an orderly release request generated at an endpoint normally operating in connection-mode. The application program can no longer receive data, but is allowed to continue sending data until the connection is fully released by issuing a TRELEASE macro instruction.

```
[ symbol ] TRELACK [ EP = endpoint_id ]
                   [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                   [ ,SYNC | ASYNC ]
                   [ ,BLOCK | NOBLOCK ]
                   [ ,ABORT | CLEAR ] ) ]
                   [ ,ECB = INTERNAL | event_control_block_addr ]
                   [ ,EXIT = tpl_exit_routine_address ]
                   [ ,MF = ( I | L | G | M | E , [ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TRELACK macro instruction executes.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

OPTCD = SHORT | LONG | EXTEND

Format attribute of the parameter list associated with this request.

OPTCD=SHORT a different control block identifier used to indicate that a subset of the TPL was generated.

OPTCD=LONG a standard, full-length TPL is generated.

OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters.

Default: SHORT if MF=I or MF operand omitted, LONG otherwise.

OPTCD = SYNC | ASYNC

Synchronization mode to use when executing the TRELACK macro instruction.

OPTCD=SYNC the request executes in synchronous mode and control is not returned to the application program until the requested macro instruction completes. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API.

OPTCD=ASYNC the request executes in asynchronous mode and control is returned immediately after scheduling the TRELACK request. The application program is responsible for issuing the TCHECK macro instruction.

Default: SYNC (synchronous mode).

OPTCD = ABORT | CLEAR

Action that the transport provider should take when the application program issues a TRELACK macro instruction at an endpoint for which a disconnect indication is already pending.

OPTCD=ABORT the request completes abnormally, and the application program must clear the pending disconnect indication with a TCLEAR macro instruction.

OPTCD=CLEAR the TRELACK request clears the disconnect indication, and the macro instruction completes normally (or conditionally) if no other errors occur.

Default: CLEAR (clear pending disconnect indication) .

ECB = INTERNAL | *event_control_block_addr*

Location of an Event Control Block (ECB) to be posted by the API when the TRELACK macro instruction associated with this TPL completes.

The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB.

The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB.

This operand is mutually exclusive with the following EXIT operand.

Default: INTERNAL (internal ECB).

EXIT = *tpl_exit_routine_address*

Address of an exit routine to schedule when the TRELACK macro associated with this TPL completes.

The TPL exit routine is scheduled only if asynchronous mode was specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB.

This operand is mutually exclusive with the previous ECB operand.

Default: Not indicated (no TPL exit routine).

MF= (I | L | G | M | E, [*tpl_address*])

Standard, list, generate, modify, or execute form of the TRELACK macro instruction.

The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used.

See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: MF=I (standard, nonreentrant form).

Completion Information

The TRELACK macro instruction completes normally when an orderly release indication is received.

- If the state of the endpoint was connected (TSCONNCT), the state is changed to release-indication-pending (TSINRLSE)
- If the state of the endpoint was not connected, the state must have been release-in-progress (TSOURLSE), and is changed to disabled (TSDSABLD) or enabled (TSENABLD), depending on the operating mode of the transport user

The connection is not released until the endpoint returns to the disabled or enabled state.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY). The conditional completion code in register zero is always zero (TCOKAY), and the TPL return code field is set accordingly. No other information is returned.

If the TRELACK macro instruction completes abnormally, the release indication (if any) remains queued. The state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TRELACK return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY		
TRFAILED	TAINTEG	TEPROTO	TEDISCON	
	TAENVRO	TESYSERR TESTOP	TESUBSYS TEUNSUPF	TEDRAIN
	TAFORMAT	TEBDFNCD TEBDEXIT	TEBDOPCD	TEBDECB
	TAPROCED	TEAMODE TENORLSE	TESTATE	TEINCMPL
	TATPLERR	TEACTIVE		
TRFATLFC	func. code	The function code loaded into register zero is invalid.		
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.		
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.		
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.		

Usage Information

The TRELACK macro instruction is used to acknowledge an orderly release indication that was generated at an endpoint. It is used in conjunction with the TRELEASE macro instruction to terminate a connection without loss of data.

An orderly release indication is generated by an orderly release protocol data unit arriving at an endpoint. When all of the data buffered at the endpoint is received by the application program, the release indication is generated. The application must issue a TRELACK macro instruction to acknowledge the indication.

- If the release was initiated by the remote transport, the application may continue sending data until a TRELEASE macro instruction is executed
- If the release was initiated by the application program, acknowledgment of the release indication causes the connection to be terminated

The orderly release procedure is primarily intended for connection-mode operation. However, if supported by the transport provider, associations established in connectionless mode can be terminated using the orderly release procedure. If the application program issues a TRELEASE macro instruction, a release indication is generated as soon as all inbound data buffered at the endpoint has been received. The orderly release service is simulated by the API, and is transparent to the transport provider.

TRELEASE

Initiate Orderly Release— The TRELEASE macro instruction is used to initiate or complete the orderly release of a connection. This macro instruction provides a graceful termination of a connection and is not immediate as is the abortive disconnect initiated with the TDISCONN macro instruction. Any data previously sent with a TSEND macro instruction is delivered.

```
[ symbol ] TRELEASE [ EP = endpoint_id ]
                    [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                    [ ,SYNC | ASYNC ] ) ]
                    [ ,ECB = INTERNAL | event_control_block_addr ]
                    [ ,EXIT = tpl_exit_routine_address ]
                    [ ,MF = ( I | L | G | M | E , [ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TRELEASE macro instruction executes.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

OPTCD = SHORT | LONG | EXTEND

Format attribute of the parameter list associated with this request.

OPTCD=SHORT a different control block identifier used to indicate that a subset of the TPL was generated.

OPTCD=LONG a standard, full-length TPL is generated.

OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters.

Default: SHORT if MF=I or MF operand omitted, LONG otherwise.

OPTCD = SYNC | ASYNC

Synchronization mode to use when executing the TRELEASE macro instruction.

OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction is complete. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API.

OPTCD=ASYNC the request executes in asynchronous mode, and returns control immediately after scheduling the TRELEASE request. The application program is responsible for issuing the TCHECK macro instruction.

Default: SYNC (synchronous mode).

ECB = INTERNAL | *event_control_block_addr*

Location of an Event Control Block (ECB) to be posted by the API when the TRELEASE macro instruction associated with this TPL completes.

The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB.

The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB.

This operand is mutually exclusive with the following EXIT operand.

Default: INTERNAL (internal ECB).

EXIT = *tpl_exit_routine_address*

Address of an exit routine to schedule when the TRELEASE macro instruction associated with this TPL completes.

The TPL exit routine is scheduled only if asynchronous mode has been specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB.

This operand is mutually exclusive with the previous ECB operand.

Default: Not indicated (no TPL exit routine).

MF = (I | L | G | M | E , [*tpl_address*])

Standard, list, generate, modify, or execute form of the TRELEASE macro instruction.

The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used.

See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: MF=I (standard, nonreentrant form).

Completion Information

The TRELEASE macro instruction completes normally when the orderly release protocol data unit is scheduled for transmission to the connected transport user.

- If the state of the endpoint was connected (TSCONNCT), the state is changed to release-in-process (TSOURLSE)
- If the state of the endpoint was not connected, the state must have been release-indication-pending (TSINRLSE), and is changed to disabled (TSDSABLD) or enabled (TSENABLD), depending on the operating mode of the application program

The connection is not released until the endpoint returns to the disabled or enabled state.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY). The conditional completion code in register zero is always zero (TCOKAY), and the TPL return code field is set accordingly. No other information is returned.

If the TRELEASE macro instruction completes abnormally, no orderly release protocol data unit is scheduled for transmission. The state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TRELEASE return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY		
TRFAILED	TAINTEG	TEPROTO	TEDISCON	
	TAENVIRO	TESYSERR TESTOP	TESUBSYS TETERM	TEDRAIN TEUNSUPF
	TAFORMAT	TEBDFNCD TEBDEXIT	TEBDOPCD	TEBDECB
	TAPROCED	TEAMODE	TESTATE	TEINCMPL
	TATPLERR	TEACTIVE		
TRFATLFC	func. code	The function code loaded into register zero is invalid.		
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.		
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.		
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.		

Usage Information

The TRELEASE macro instruction is used to initiate or complete the orderly release of a connection. It differs from the TDISCONN macro instruction in that it is not immediate, and the connection is not released until both transport users agree to do so. Any unsent data buffered at the endpoint is forwarded to the peer transport user. The TRELEASE macro instruction does not complete until all pending TSEND macro instructions complete.

The orderly release procedure requires both transport users to request orderly release before the connection is terminated. If the application program initiates orderly release by executing a TRELEASE macro instruction, all buffered data is forwarded to the peer transport user, followed by an orderly release request. The application program should then continue receiving data until an orderly release indication is received. Any attempt to send data through an endpoint in the release-in-progress state completes abnormally.

If the peer transport user initiates an orderly release, a release indication is generated and presented to the application program. The application program should cease receiving data, and execute a TRELACK macro instruction to acknowledge the release indication. The application program can continue sending data until transmission is complete, and then a TRELEASE macro instruction should be issued to complete termination of the connection.

Orderly release may not be supported by the transport provider, and should be requested when the endpoint is opened (see TOPEN). If not requested in advance, the TINFO macro instruction should be issued to determine if orderly release is supported. If not, the application program should use a session protocol, or some other method to terminate the session with the peer transport user, to avoid losing data that may be discarded by an abortive disconnect.

The TRELEASE macro instruction is generally executed at endpoints operating in connection mode. If an association has been established for an endpoint operating in connectionless mode, the API simulates the orderly release procedure, and generates a release indication after all data has been sent to the peer transport user.

Note: There is no guarantee that the data was received. This procedure is transparent to the transport provider.

TRETRACT

Retract Pending Listen Request— The TRETRACT macro instruction is used to retract a pending listen initiated with the TLISTEN macro instruction. Any outstanding listen is forced to complete abnormally, and the state of the endpoint is as if the TLISTEN macro instruction had not been executed at all.

```
[ symbol ] TRETRACT [ EP = endpoint_id ]
                    [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                    [ ,SYNC | ASYNC ] ) ]
                    [ ,ECB = INTERNAL | event_control_block_addr ]
                    [ ,EXIT = tpl_exit_routine_address ]
                    [ ,MF = ( I | L | G | M | E , [ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TRETRACT macro instruction executes.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

OPTCD = SHORT | LONG | EXTEND

Fformat attribute of the parameter list associated with this request.

OPTCD=SHORT a different control block identifier used to indicate that a subset of the TPL was generated.

OPTCD=LONG a standard, full-length TPL is generated.

OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters.

Default: SHORT if MF=I, LONG otherwise.

OPTCD = SYNC | ASYNC

Synchronization mode to use when executing the TRETRACT macro instruction.

OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction completes. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API.

OPTCD=ASYNC the request executes in asynchronous mode, and returns control immediately after scheduling the TRETRACT request. The application program is responsible for issuing the TCHECK macro instruction.

Default: SYNC (synchronous mode).

ECB = INTERNAL | *Event_control_block_addr*

Location of an Event Control Block (ECB) to be posted by the API when the TRETRACT macro instruction associated with this TPL completes.

The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB.

The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB.

This operand is mutually exclusive with the following EXIT operand.

Default: INTERNAL (internal ECB).

EXIT = *tpl_exit_routine_address*

Address of an exit routine to schedule when the TRETRACT macro instruction associated with this TPL completes.

The TPL exit routine is scheduled only if asynchronous mode was specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB.

This operand is mutually exclusive with the previous ECB operand.

Default: Not indicated (no TPL exit routine).

MF = (I | L | G | M | E , [*tpl_address*])

Standard, list, generate, modify, or execute form of the TRETRACT macro instruction.

The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used.

See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: MF=I (standard, nonreentrant form).

Completion Information

The TRETRACT macro instruction completes normally when a TLISTEN request, which was pending at the endpoint, is prematurely terminated. The pending TLISTEN request completes abnormally, and the specific error code is set to TERETRCT to indicate that a TRETRACT macro instruction forced its completion. The state of the endpoint is unchanged, and is as if the TLISTEN macro instruction had not been executed. If no connect indications were pending at the time the listen was retracted, the endpoint is left in the enabled (TSENABLD) state. Otherwise, it is left in the connect-indication-pending (TSINCONN) state.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY). The conditional completion code in register zero is always zero (TCOKAY), and the TPL return code field is set accordingly. No other information is returned.

If no TLISTEN request was pending, perhaps because it had already completed, the TRETRACT macro instruction completes abnormally. The general return code in register 15, and the recovery action code in register zero, indicate the nature of the failure. The TPL return code field can contain a specific error code that identifies the particular error.

Return Codes

The following table lists the symbolic names for the TRETRACT return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY		
TRFAILED	TAEXCPTN	TENOLSTN		
	TAENVIRO	TESYSERR	TESUBSYS	TEDRAIN
		TESTOP	TETERM	
	TAFORMAT	TEBDFNCD TEBDEXIT	TEBDOPCD	TEBDECB
	TAPROCED	TEAMODE	TESTATE	TEINCMPL

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation
	TATPLERR	TEACTIVE
TRFATLFC	func. code	The function code loaded into register zero is invalid.
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.

Usage Information

The TRETRACT macro instruction can be used to retract a pending TLISTEN macro instruction by forcing it to complete abnormally. Once the TLISTEN request completes, the protocol address can be unbound from the endpoint. Until then, the endpoint can continue receiving connect indications. TRETRACT provides an alternative to closing the endpoint in order to prevent it from listening for connect indications. If any connect indications are pending at the endpoint, they remain pending, and must be accepted or rejected before the protocol address may be unbound.

There is no guarantee that the TLISTEN request can be retracted before it would be completed normally by an incoming connection request. If this happens, the TRETRACT function does not find any record of a pending TLISTEN request, and completes the TRETRACT macro instruction abnormally. If the TRETRACT macro instruction was issued too late to prevent TLISTEN from completing normally, the pending connect indication must be processed as usual by accepting or rejecting it with the appropriate macro instruction.

TSEND

Send Normal or Expedited Data on a Connection— The TSEND macro instruction is used to send normal or expedited data to the peer transport user connected to an endpoint. TSEND is normally used to send data on an endpoint operating in connection mode, but when supported by the transport provider, datagrams may be sent on an endpoint operating in connectionless-mode if the application program has established an association with the transport user.

```
[ symbol ] TSEND [ EP = endpoint_id ]
                 [ ,DALEN = user_data_length ]
                 [ ,DABUF = user_data_address ]
                 [ ,DAALET = user_address_alet ]
                 [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                              [ ,SYNC | ASYNC ]
                              [ ,MORE | NOMORE ]
                              [ ,NORMAL | EXPEDITE ]
                              [ ,EOM | NOTEOM ]
                              [ ,DIRECT | INDIR ]
                              [ ,BLOCK | NOBLOCK ]
                              [ ,MBUF | NOMBUF ] ) ]
                  [ ,ECB = INTERNAL | event_control_block_addr ]
                  [ ,EXIT = tpl_exit_routine_address ]
                  [ ,MF = ( I | L | G | M | E,[ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TSEND macro instruction executes.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

DALEN = *user_data_length*

Length (in bytes) of a data storage area or an indirect data vector identified by the DABUF operand.

If the data mode is direct, the amount of user data to be sent is equal to the length of the storage area.

If the data mode is indirect, the total amount of user data is equal to the sum of all data segments identified by the data vector.

In either case, the total amount of user data must not exceed the limit supported by the transport provider. This limit can be obtained with the TINFO macro instruction. A length of zero indicates there is no user data to be sent.

Default: Zero (no user data).

DABUF = *user_data_address*

Address of user data to be sent to the connected (or associated) transport user.

If the data mode is direct, the value specified is the address of the storage area containing the user data.

If the data mode is indirect, the value specified must be the address of an indirect data vector, and each element of the vector must have been initialized to point to an individual segment of user data.

If no data is available, the length as indicated by the DALEN operand should be zero. The content of all user data is application-dependent, and is not interpreted by the API or the transport provider. The storage area can be aligned on any boundary convenient for the application program.

Default: Zero (no user data storage area).

DAALET = user_address_alet

Access List Entry Token (ALET) used in access register (AR) mode when referencing the storage specified by the DABUF parameter.

The DAALET value must be an ALET contained in the Dispatchable Unit Access List (DUAL) of the caller. The DAALET parameter can be used only if OPTCD=EXTEND is also specified.

Default: Zero (the storage is contained in the address space of the caller).

OPTCD = SHORT | LONG | EXTEND

Format attribute of the parameter list associated with this request.

OPTCD=SHORT a different control block identifier used to indicate that a subset of the TPL was generated.

OPTCD=LONG a standard, full-length TPL is generated.

OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters.

Default: SHORT if MF=I or MF operand omitted, LONG otherwise.

OPTCD = SYNC | ASYNC	Synchronization mode to use when executing the TSEND macro instruction. OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction completes. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API. OPTCD=ASYNC the request executes in asynchronous mode, and returns control immediately after scheduling the TSEND request. The application program is responsible for issuing the TCHECK macro instruction. Default: SYNC (synchronous mode).
OPTCD = MORE | NOMORE	Indicates whether the application program intends to immediately send more data, or intends to pause momentarily until it has more data to send. OPTCD=MORE the application program expects to immediately issue another TSEND macro instruction at the same endpoint. OPTCD=NOMORE the application program has no more data to send, but intends to leave the connection established, and may resume sending data later. The interpretation of this option code by the transport provider is protocol dependent. The intent is that the transport provider uses this information to augment its packet algorithm, and deduce when unsent data must be forwarded on the connection. Not all connection-mode transport providers are required to interpret this option code, but all are required to accept it. If the TSEND macro instruction is executed in synchronous mode, OPTCD=MORE is ignored. No implication is drawn about message boundaries by the indication of OPTCD=NOMORE. If the underlying transport protocol can preserve logical boundaries within the data stream, then such boundaries should be indicated with OPTCD=EOM. Default: NOMORE (send data immediately).

OPTCD = NORMAL | EXPEDITE

Indicates whether the user data should be sent as normal or expedited data.

OPTCD=NORMAL the data associated with this request is sent as normal data.

OPTCD=EXPEDITE the data is to be sent as expedited data.

The distinction between normal and expedited data is left to the interpretation of the transport provider.

Default: NORMAL (send data as normal data).

OPTCD = EOM | NOTEOM

Indicates whether the data associated with this request is a complete message, or is continued with one or more subsequent macro instructions.

OPTCD=EOM the last byte of data corresponds to the end of the message or datagram.

OPTCD=NOTEOM the end of the message or datagram does not occur with this request, and is continued with at least one more TSEND or TSENDTO macro instruction.

Although all transport providers supplying connection-mode service are required to accept this option code, only those that can preserve logical boundaries in the data stream are required to interpret it. Generally, these providers support the concept of a transport service data unit to delineate such boundaries.

If the transport provider does not preserve logical boundaries, this option code is ignored. The setting of this option code implies nothing about how the user data is broken down into packets by the underlying protocol for sending to the peer transport user.

Transport providers supplying connectionless-mode service are not required to accept this option code. Those that do interpret OPTCD=NOTEOM to mean the datagram is continued with another TSEND macro instruction. In this case, a datagram is synonymous with TSDU, except that implementation is purely a local concern.

Default: EOM (end of message or datagram).

OPTCD = DIRECT | INDIR

Format of the user data parameter.

OPTCD=DIRECT the DABUF and DALEN operands identify a storage area into which data should be received directly.

OPTCD=INDIR the storage area identified by these operands contains an indirect data vector. An indirect data vector consists of a list of address-length pairs, with each element identifying a separate segment of non-contiguous storage. In this case, DABUF is the address of the first element in the list, and DALEN is the total length of the list.

The length of the vector must be a multiple of eight, and the total amount of data that can be received is the sum of the lengths of each data segment.

Default: DIRECT (send directly from data area).

OPTCD = MBUF | NOMBUF

DABUF parameter is the address of a TCPaccess MBUF, rather than a data buffer or indirect buffer list.

OPTCD=MBUF the DALEN parameter is ignored and the length of the data is determined from fields within the MBUF structure.

OPTCD=NOMBUF the DABUF parameter is processed normally.

Note: OPTCD=MBUF is intended only for applications internal to TCPaccess and is used to improve performance.

Default: NOMBUF.

OPTCD = BLOCK | NOBLOCK

OPTCD=NOBLOCK may be used with endpoints opened with MODE=SOCKETS.

This option is ignored for TLI-mode endpoints, which always block until the sent data is acknowledged. Normally, socket-mode endpoints do not block. However, if the amount of send data exceeds the amount of available buffer space, the TSEND request blocks by default until buffer space becomes available.

OPTCD=NOBLOCK can be used in this case to prevent the endpoint from becoming blocked. When this occurs, only the amount of data for which there is space is sent.

Buffer space is limited by configuration parameters and TOPTION negotiation.

If the option is OPTCD=NOBLOCK, then the TSEND sends either the amount of available space in the send buffer or the amount of data that was requested by the TSEND, whichever is less.

Default: BLOCK.

ECB = INTERNAL | *event_control_block_addr*

Location of an Event Control Block (ECB) to be posted by the API when the TSEND macro instruction associated with this TPL completes.

The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB.

The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB.

This operand is mutually exclusive with the following EXIT operand.

Default: INTERNAL (internal ECB).

EXIT = *tpl_exit_routine_address*	Address of an exit routine to schedule when the TSEND macro instruction associated with this TPL completes. The TPL exit routine is scheduled only if asynchronous mode was specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB. This operand is mutually exclusive with the previous ECB operand. Default: Not indicated (no TPL exit routine).
MF = (I | L | G | M | E , [*tpl_address*])	Standard, list, generate, modify, or execute form of the TSEND macro instruction. The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used. See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters. Default: MF=I (standard, nonreentrant form).

Completion Information

The TSEND macro instruction completes normally when the data from the application program's storage area is moved, and forwarded to the transport provider for sending to the connected (or associated) transport user. For MODE=SOCKETS, the count of the data bytes sent is returned in the TPLCOUNT field.

Normal completion of the TSEND macro implies nothing in regard to when the data is sent to the peer transport user, and should be interpreted to mean that the transport provider has taken custody of the user data, and the storage area provided by the application program can be reused by another TSEND macro instruction. If OPTCD=NOTEOM was indicated, no assumption should be made (unless the endpoint is operating in connectionless mode) about previous fragments of the current TSDU not being sent. If OPTCD=NOMORE was indicated, the transport provider is normally coerced into sending any buffered data, but this may not occur synchronously with the completion of the macro instruction.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY). The conditional completion code in register zero is always zero (TCOKAY), and the TPL return code field is set accordingly. No other information is returned.

If the TSEND macro instruction completes abnormally, no user data is sent to the peer transport user. The state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TSEND return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY		
TRFAILED	TAEXCPTN	TENOBLOK		
	TAINTEG	TEPROTO	TEDISCON	
	TAENVIRO	TESYSERR TESTOP	TESUBSYS TETERM	TEDRAIN TEUNSUPF
	TAFORMAT	TEBDFNCD TEBDEXIT	TEBDOPCD TEBDDATA	TEBDECB
	TAPROCED	TEAMODE TEBUFOVR	TESTATE	TEREQOVR
	TATPLERR	TEACTIVE		
TRFATLFC	func. code	The function code loaded into register zero is invalid.		
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.		

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.

Usage Information

The TSEND macro instruction is used to send normal or expedited data through an endpoint. The data may be part of a byte stream or message stream being sent over a connection, or part (or all) of a datagram to be sent via an association to a peer transport user.

- If the transport service type is a connection-mode byte stream, data is moved from the application program's storage areas to storage areas maintained by the transport provider, broken down into packets, and sent to the connected transport user. Logical boundaries are not preserved in the data stream. The data is delivered to the peer transport user in the precise order in which it was sent, but may be fragmented in an entirely different manner. The EOM and NOTEOM indications set in the TPL are ignored.
- If the transport service type is a connection-mode message stream, data is processed in a similar manner. However, the EOM and NOTEOM indications set in the TPL are used to delineate the boundaries of transport service data units. When the data is delivered to the peer transport user, the continuation or end of a TSDU is similarly indicated. The concept of a TSDU implies nothing about how the underlying protocol creates data packets for transmission. The maximum size of a TSDU can be determined by issuing a TINFO macro instruction.

Transport providers operating in connectionless mode are not required to accept or interpret the EOM and NOTEOM indications. If interpreted, the concept of a message (or TSDU) is synonymous with a datagram, and the significance of the EOM and NOTEOM indications is only a local phenomenon. The intent is to provide a mechanism for the application program to send a large message as multiple, noncontiguous fragments. If the transport provider supports this option code, it is required to buffer the fragments and send the message as a single datagram.

Data is not necessarily broken down into packets and sent each time a TSEND macro instruction is issued, or when a message boundary is indicated. In fact, the transport provider may intentionally delay sending data as the result of performance optimization or congestion avoidance algorithms. Generally, a continual flow of data generated by the sender causes data to be forwarded. However, in interactive applications, the transport provider may need to be coerced into forwarding any buffered data.

The MORE and NOMORE indications control this process.

- If MORE is indicated, the application intends to immediately send more data, and the transport provider is free to delay sending any data associated with the current request
- If NOMORE is indicated, the application has no more data to send, and all buffered data should be forwarded to the peer transport user

For example, if TCP is the underlying protocol, the NOMORE indication would cause the PUSH flag to be set.

NOMORE is ignored when the synchronization mode is ASYNC.

If the transport provider supports expedited data, the TSEND macro instruction is also used to send it. The NORMAL and EXPEDITE indicators are used to distinguish normal and expedited data. The resulting actions of the transport provider are protocol-dependent. Some transport providers support the concept of an expedited transport service data unit, and others support the concept of expedited data within a data stream without logical boundaries. The EOM and NOTEOM indications apply to an expedited TSDU in the same way they apply to a normal TSDU. The form of expedited data supported by the transport provider, and the maximum size of an ETSDU, can also be determined with the TINFO macro instruction.

The OPTCD field of the TPL is used by the application program to set indicators that can be tested by the API data transfer routines. The indicators set by the TSEND macro instruction correspond to the indicators returned by the TRECV macro instruction. These indicators are located in the function-specific option code field mapped by the TPL DSECT as TPLOPCD2.

These bits are used by the TSEND macro instruction:

- If set, the TONOTEOM bit corresponds to OPTCD=NOTEOM, and indicates that another macro instruction is issued to send the continuation of the TSDU or datagram
- If not set, the TONOTEOM corresponds to OPTCD=EOM, and indicates the last byte of data moved from the storage area corresponds to the end of the TSDU or datagram
- If set, the TOMORE bit corresponds to OPTCD=MORE, and indicates that the application program intends to immediately issue another TSEND macro instruction to send more data. The data is not necessarily part of the current message
- If not set, the TOMORE bit corresponds to OPTCD=NOMORE, and indicates that the transport provider should forward all data buffered at the endpoint to the peer transport user
- If set, the TOEXPDTE bit corresponds to OPTCD=EXPEDITE, and indicates that the data contained in the storage area should be sent as expedited data
- If not set, the TOEXPDTE bit corresponds to OPTCD=NORMAL, and indicates that the data should be sent as normal data. If the transport provider supports the concept of an expedited transport service data unit (ETSDU), TONOTEOM is used to mark the continuation of the ETSDU

User data may be provided in a simple, contiguous segment of storage, or in a set of non-contiguous segments indirectly addressed via a data vector.

The storage area provided by the application program can be a simple, contiguous segment of storage, or a set of non-contiguous segments indirectly addressed via a data vector.

If the option is:

- OPTCD=DIRECT user data must be contained in the storage area identified by the DABUF and DALEN operands.
- OPTCD=INDIR the DABUF and DALEN operands identify a storage area initialized with the addresses and lengths of non-contiguous storage segments containing the user data. The total amount of data to be transferred is the sum of the lengths of the individual segments. The total length must not exceed the maximum size of the interface data unit supported by the transport provider, or the maximum size of a transport service data unit. Upon completion of the TSEND request, the length of the indirect data vector is updated to reflect the actual amount of data transferred.

Each entry in an indirect data vector consists of a fullword address followed by a fullword length. If the length is zero, the entry is ignored. If the length is nonzero, the address must reference a valid storage area containing user data, and can be aligned on any boundary convenient for the application program. The length of the vector is used to determine the number of entries in the list.

Unlike most other macro instructions, multiple TSEND macro instructions can be issued without waiting for the first to complete. However, each macro instruction requires its own TPL. The maximum number that can be issued before one must complete is the API variable that can be modified by the TOPTION macro instruction. The default value is set when the API is installed. TSEND macro instructions are completed in the order in which they are issued.

Data sent with the TSEND macro instruction is buffered in the API address space before it is forwarded to the transport provider. The total amount of send buffering allocated for an endpoint is also the API option.

User data is application-dependent, and is not interpreted by the API or the transport provider. The maximum amount that can be sent with a single TSEND request can be determined by issuing a TINFO macro instruction.

Data Transfer Modes

TSEND handles data transfer according to the transfer mode specified on the TOPEN macro. TLI or SOCKET may be specified; TLI is the default.

TLI Mode

TLI is the standard data transfer mode for TCPaccess. For TLI mode, specify MODE=TLI on the TOPEN macro, or allow it to default.

- A TSEND request in TLI mode completes when all of the data is acknowledged by the remote TCP. The amount of data that can be sent is subject to the limits set by the LSEND and LTSND values.
- A TLI mode TSEND sends either all of the data, or none of it.
- Data contained in the application program's storage area is moved into the API address space when the TSEND macro instruction is accepted. Therefore, if the TSEND macro instruction is executed in asynchronous mode, the application program can reuse the storage area after the macro instruction completes but before the request is completed. In fact, the storage area may be used to send more data as long as an inactive TPL is available.
- TPLCOUNT is cleared to zero.

Socket Mode

For socket mode, specify MODE=SOCKET on the TOPEN macro.

- A TSEND request in socket mode completes when all of the data that will be sent is passed to the local transport provider (for example, TCP or UDP).
- The amount of data that is actually sent for TSEND depends on whether OPTCD=BLOCK or OPTCD=NOBLOCK is used. In either case, the amount of sent data is returned in the TPLCOUNT field of the TPL.

Note: OPTCD=BLOCK | NOBLOCK applies only to MODE=SOCKETS endpoints.

- OPTCD=BLOCK: All of the data in the send request is sent.
- OPTCD=NOBLOCK: All, some, or none of the data may be sent. The amount of data that is actually sent depends on the space available in the current send buffer.
- Data contained in the application program's storage is moved into the API address space when it is scheduled for transmission. Therefore, if the TSEND macro instruction is issued in asynchronous mode, the application **cannot** reuse the storage area until the TSEND request completes.

TSENDTO

Send a Datagram— The TSENDTO macro instruction is used to send datagrams to a remote transport user through an endpoint operating in connectionless-mode. The user data, the remote protocol address of the destination, and any options associated with the datagram are provided by the application program.

```
[ symbol ] TSENDTO [ EP = endpoint_id ]
                   [ ,ADLEN = protocol_address_length ]
                   [ ,ADBUF = protocol_address_address ]
                   [ ,ADALET = protocol_address_alet ]
                   [ ,DALEN = user_data_length ]
                   [ ,DABUF = user_data_address ]
                   [ ,DAALET = user_data_alet ]
                   [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                                [ ,SYNC | ASYNC ]
                                [ ,DIRECT | INDIR ]
                                [ ,MBUF | NOMBUF ]
                                [ ,BLOCK | NOBLOCK ] ) ]
                   [ ,ECB = INTERNAL | event_control_block_addr ]
                   [ ,EXIT = tpl_exit_routine_address ]
                   [ ,MF = ( I | L | G | M | E,[ tpl_address ] ) ]
```

EP = *endpoint_id*

Specifies the endpoint at which the TSENDTO macro instruction executes.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

ADLEN = *protocol_address_length*

Indicates the length (in bytes) of the protocol address contained in the storage area identified by the ADBUF operand.

A length of zero is invalid, and causes the request to be abnormally completed.

Default: Zero (no protocol address).

ADBUF = *protocol_address_address*

Address of a storage area containing the protocol address of the destination transport user that will receive the datagram.

The length of the protocol address is designated by the ADLEN operand. The protocol address can be aligned on any boundary convenient for the application program.

Default: Zero (no protocol address storage area).

ADALET = *protocol_address_alet*

Access List Entry Token (ALET) used in access register (AR) mode when referencing the storage specified by the ADBUF parameter.

The ADALET value must be an ALET contained in the Dispatchable Unit Access List (DUAL) of the caller. The ADALET parameter can only be used if OPTCD=EXTEND is also specified.

Default: Zero (the storage is contained in the address space of the caller).

DALEN = *user_data_length*

Length (in bytes) of a data storage area or an indirect data vector identified by the DABUF operand.

If the data mode is direct, the amount of user data sent is equal to the length of the storage area.

If the data mode is indirect, the total amount of user data is equal to the sum of all data segments identified by the data vector.

In either case, the total amount of user data must not exceed the limit supported by the transport provider. This limit can be obtained with the TINFO macro instruction. A length of zero indicates there is no user data to be sent.

Default: Zero (no user data).

DABUF = *user_data_address*

Address of user data to be sent to the specified transport user.

If the data mode is direct, the value specified is the address of the storage area containing the user data.

If the data mode is indirect, the value specified must be the address of an indirect data vector, and each element of the vector must have been initialized to point to an individual segment of user data.

If no data is available, the length as indicated by the DALEN operand should be zero. The content of all user data is application-dependent, and is not interpreted by the API or the transport provider. The storage area can be aligned on any boundary convenient for the application program.

Default: Zero (no user data storage area).

DAALET = *user_data_alet*

Access List Entry Token (ALET) used in access register (AR) mode when referencing the storage specified by the DABUF parameter.

The DAALET value must be an ALET contained in the Dispatchable Unit Access List (DUAL) of the caller. The DAALET parameter can only be used if OPTCD=EXTEND is also specified.

Default: Zero (the storage is contained in the address space of the caller).

OPTCD = SHORT | LONG | EXTEND

Format attribute of the parameter list associated with this request.

OPTCD=SHORT a different control block identifier is used to indicate that a subset of the TPL has been generated.

OPTCD=LONG a standard, full-length TPL is generated.

OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters.

Default: SHORT if MF=I or MF operand omitted, LONG otherwise.

OPTCD = SYNC | ASYNC

Synchronization mode to use when executing the TSENDTO macro instruction.

OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction is complete. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API.

OPTCD=ASYNC the request executes in asynchronous mode, and returns control immediately after scheduling the TSENDTO request. The application program is responsible for issuing the TCHECK macro instruction.

Default: SYNC (synchronous mode).

OPTCD = DIRECT | INDIR

Format of the user data parameter.

OPTCD=DIRECT the DABUF and DALEN operands identify a storage area into which data should be received directly.

OPTCD=INDIR the storage area identified by these operands contains an indirect data vector.

An indirect data vector consists of a list of address-length pairs, with each element identifying a separate segment of non-contiguous storage. In this case, DABUF is the address of the first element in the list, and DALEN is the total length of the list. The length of the vector must be a multiple of eight, and the total amount of data that can be received is the sum of the lengths of each data segment.

Default: DIRECT (send directly from data area).

OPTCD = MBUF | NOMBUF

DABUF parameter is the address of a TCPaccess MBUF, rather than a data buffer or indirect buffer list.

OPTCD=MBUF the DALEN parameter is ignored and the length of the data is determined from fields within the MBUF structure.

Note: OPTCD=MBUF is intended only for applications internal to TCPaccess and is used to improve performance.

OPTCD=NOMBUF the DABUF parameter is processed normally.

Default: NOMBUF.

OPTCD = BLOCK | NOBLOCK

OPTCD=NOBLOCK may be used with endpoints opened with MODE=SOCKETS.

This option is ignored for TLI-mode endpoints, which always block until the data is passed to the local network. Normally, socket-mode endpoints do not block. However, if the amount of available buffer space is exceeded, the TSENDTO request blocks by default until buffer space becomes available. OPTCD=NOBLOCK can be used in this case to prevent the endpoint from becoming blocked.

Buffer space is limited by configuration parameters and TOPTION negotiation.

Default: BLOCK.

ECB = INTERNAL | *event_control_block_addr*	Location of an Event Control Block (ECB) to be posted by the API when the TSENDTO macro instruction associated with this TPL completes. The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB. The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB. This operand is mutually exclusive with the following EXIT operand. Default: INTERNAL (internal ECB).
EXIT = *tpl_exit_routine_address*	Address of the exit routine to schedule when the TSENDTO macro instruction associated with this TPL completes. The TPL exit routine is scheduled only if asynchronous mode was specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB. This operand is mutually exclusive with the previous ECB operand. Default: Not indicated (no TPL exit routine).
MF = (I | L | G | M | E , [*tpl_address*])	Standard, list, generate, modify, or execute form of the TSENDTO macro instruction. The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used. See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters. Default: MF=I (standard, nonreentrant form).

Completion Information

The TSENDTO macro instruction completes normally (or conditionally) when the datagram is moved from the application program's storage area, and is forwarded to the transport provider for sending to the destination transport user.

Normal completion of the TSENDTO macro instruction implies nothing in regard to when the datagram is actually sent, and should only be interpreted to mean that the transport provider has taken custody of the user data, and the storage area provided by the application program can be reused by another TSENDTO macro instruction.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY), and a conditional completion code is returned in register zero. The TPL return code field is set accordingly. No other information is returned.

If the TSENDTO macro instruction completes abnormally, the datagram is not sent to the destination transport user. The state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TSENDTO return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY		
TRFAILED	TAEXCPTN	TENONEGO		
	TAINTEG	TEPROTO		
	TAENVIRO	TESYSERR TESTOP TEUNSUPF	TESUBSYS TETERM	TEDRAIN TEUNSUPO
	TAFORMAT	TEBDFNCD	TEBDOPCD	TEBDECB
		TEBDEXIT	TEBDDATA	TEBDOPTN

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation
	TAPROCED	TEAMODE TESTATE TEREQOVR TEBUFOVR
	TATPLERR	TEACTIVE
TRFATLFC	func. code	The function code loaded into register zero is invalid.
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.

Usage Information

The TSENDTO macro sends a datagram through an endpoint operating in connectionless mode. In addition to user data comprising the datagram, the application program provides the protocol address of the destination transport user and protocol options associated with the datagram. The datagram is transmitted as a single, contiguous unit of data and must be provided to the transport provider in its entirety.

The datagram may be provided as a simple, contiguous segment of storage, or as a set of non-contiguous segments indirectly addressed via a data vector.

If the option is:

- OPTCD=DIRECT a complete datagram must be contained in the storage area identified by the DABUF and DALEN operands.
- OPTCD=INDIR—DABUF and DALEN identify a storage area initialized with the addresses and lengths of non-contiguous storage segments containing the datagram. The total amount of data to be transferred is the sum of the lengths of the individual segments. The total length must not exceed the maximum size of the interface data unit supported by the transport provider, or the maximum size of a transport service data unit. Upon completion of the TSENDTO request, the length of the indirect data vector is updated to reflect the actual amount of data transferred.

Each entry in an indirect data vector consists of a fullword address followed by a fullword length. If the length is zero, the entry is ignored. If the length is nonzero, the address must reference a valid storage area containing user data, and may be aligned on any boundary convenient for the application program. The length of the vector determines the number of entries in the list.

Unlike most other macro instructions, multiple TSENDTO macro instructions can be issued without waiting for the first to complete. However, each macro requires its own TPL. The maximum number that can be issued before one must complete is an API variable that can be modified by the TOPTION macro. The default value is set when the API is installed. TSENDTO macros are completed in the order in which they are issued.

Datagrams sent with the TSENDTO macro are buffered in the API address space before they are forwarded to the transport provider. The total amount of send buffering allocated for an endpoint is also the API option.

Data contained in the application program's storage area is moved into the API address space when the TSENDTO macro is accepted. Therefore, if the TSENDTO macro instruction is executed in asynchronous mode, the application program can reuse the storage area before the macro instruction completes. In fact, the storage area can be used to send another datagram as long as an inactive TPL is available. The content of a datagram is application-dependent, and is not interpreted by the API or the transport provider. The maximum amount that can be sent with a single TSENDTO request can be determined by issuing a TINFO macro instruction.

TSTATE

Test TPL and Return Endpoint State— The TSTATE macro instruction is used to acquire the current state of an endpoint. Since the TSTATE macro instruction is TPL-based like most other API macro instructions but does not modify any fields in the TPL, it can also be used to determine the active or inactive state of a TPL.

```
[ symbol ] TSTATE MF = ( E, tpl_address )
```

MF = (E, *tpl_address*)

Execute form of the TSTATE macro instruction.

The second sublist operand, *tpl_address*, specifies the address of the TPL identifying the entry point.

See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: None (must be coded as indicated).

Completion Information

If the TSTATE macro instruction completes normally, the general return code is set to zero (TROKAY), and a fullword of state information is returned in register zero. The TPL return code field is not modified, and no other information is returned.

If the TSTATE macro instruction completes abnormally, the general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. If the general return code is TRFAILED, the TPL was active, and the recovery action code indicates whether or not the active request was posted complete. Otherwise, a fatal error occurred. The TPL return code field is not updated, and the state of the endpoint is unchanged.

Note: The SYNAD or LERAD exit routines are not entered when the TSTATE macro instruction completes abnormally.

Return Codes

The following table lists the symbolic names for the TSTATE return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

Register 15	Register 0	Explanation
TROKAY	Endpoint State	The TPL is inactive, and the TPLEPID field designates a valid endpoint. The state of the endpoint is returned in register zero.
TRFAILED	TAEXCPTN	The TPL is active, and the requested operation has been posted complete. A TCHECK macro instruction does not suspend the issuing task.
TRFAILED	TATPLERR	The TPL is active, and the requested operation has not been posted complete. A TCHECK macro instruction may cause the issuing task to be suspended.
TRFATLFC	func. code	The function code loaded into register zero is invalid.
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.

Note: Conditional or specific error codes are not applicable to the TSTATE macro.

Usage Information

The TSTATE macro instruction is used to determine the state of an endpoint, and implicitly, the state of a TPL that references it.

The TSTATE macro instruction is TPL-based, and therefore requires a TPL for making the request. If the TPL is valid and inactive, a fullword of state information is returned to the application program in register zero. The TPL is unchanged. The information returned is called a Transport Endpoint State Word (TSW), and is mapped by the TSW DSECT.

The state word consists of two components:

- A halfword containing status bits representing pending activity on the endpoint
- A halfword state value that represents the current state of the endpoint

This state information is standard for all endpoints, and all transport providers. However, not all states are valid for a particular provider. For example, if a transport provider does not support orderly release of a connection, the endpoint can never acquire the release-in-progress state.

Endpoint	State	State Description
TSCLOSED	Closed	The state of an endpoint before it is opened with a TOPEN macro instruction. By definition, an endpoint that exits cannot be in the closed state. Any the API macro instruction executed at an endpoint in the closed state results in a fatal error. An endpoint returns to the closed state when it is closed by a TCLOSE macro instruction.
TSOPENED	Opened	The state of an endpoint immediately after being opened with a TOPEN macro instruction. An endpoint in the opened state is not associated with any local protocol address, and cannot receive inbound or outbound connection requests. An endpoint returns to the opened state after being unbound with a TUNBIND macro instruction.

Endpoint	State	State Description
TSDSABLD	Disabled	The state of an endpoint immediately after a local protocol address has been bound with a TBIND macro instruction, and before it is enabled to receive connect indications. An endpoint in the disabled state and operating in connectionless mode is ready to send or receive datagrams. An endpoint in the disabled state and operating in connection mode is ready to initiate a connection request. A client-mode endpoint returns to the disabled state after a connection is released.
TSINCONN	Enabled	The state of an endpoint after a local protocol address is bound, and a nonzero value for QLSTN is specified in the TBIND macro instruction. An endpoint in the enabled state cannot operate in connectionless mode. An endpoint in the enabled state and operating in connection mode is ready to receive connect indications. A server-mode endpoint returns to the enabled state after a connect indication is accepted (multithreaded mode) or rejected, or when the connection is released.

Endpoint	State	State Description
TSINCONN	Connect-indication-pending	The state of an endpoint when one or more connect indications are received with the TLISTEN macro instruction that have not been accepted or rejected by the application program. The endpoint remains in the connect-indication-pending state as long as at least one indication remains pending, even though some have been accepted or rejected.
TSOUCONN	Connection-in-progress	The state of an endpoint when a TCONNECT macro instruction has been executed, and a connect confirmation has not been received by the application program. The endpoint remains in the connect-in-progress state until a TCONFIRM macro instruction is executed to receive the connect confirmation.
TSCONNCT	Connected	The state of an endpoint after a connect indication is accepted with a TACCEPT macro instruction, or a confirm indication is received with a TCONFIRM macro instruction. In single-threaded mode, the endpoint that received the connect indication enters the connected state; in multithreaded mode, the connection is accepted to a disabled endpoint, causing it to enter the connected state. An endpoint in the connected state and operating in connection mode is ready to send and receive data. An endpoint operating in connectionless mode enters the connected state when an association is established, and becomes ready to send and receive datagrams with the TSEND and TRECV macro instructions.
TSINRLSE	Release-indication-pending	The state of an endpoint that was connected after a TRELACK macro instruction is executed to acknowledge an orderly release indication. The application program may continue sending data through the endpoint.

Endpoint	State	State Description
		The endpoint remains in the release-indication-pending state until a TRELEASE macro instruction is executed, at which time it returns to the enabled or disabled state.
TSOURLSE	Release-in-progress	The state of an endpoint that was connected after a TRELEASE macro instruction is executed. The application program may continue receiving data arriving at the endpoint. The endpoint remains in the release-in-progress state until a release indication is acknowledged with the TRELACK macro instruction, at which time it returns to the enabled or disabled state.

State transitions occur when the macro instruction causing the transition completes normally and the request is posted complete, either by posting the ECB associated with the TPL, or entering the TPL exit routine. A state transition **never** occurs when a macro instruction completes abnormally.

If the TPL is active, the TSTATE macro instruction completes abnormally. The recovery action code returned in register zero can be tested to determine if the active request was posted complete. If it has, a TCHECK macro instruction can be executed at the TPL without causing a system WAIT to be issued by the API. Thus, the TSTATE macro instruction can be used to poll the TPL to determine when it is safe to issue a TCHECK macro instruction without suspending the issuing task.

A TSTATE macro instruction can be executed at any endpoint using any TPL (active or inactive) without affecting the state of the endpoint, or modifying the TPL.

TUNBIND

Unbind Protocol Address from Endpoint— The TUNBIND macro instruction is used to disable an endpoint and unbind the local protocol address that was previously bound to it with a TBIND macro instruction. Once disabled, the endpoint can no longer receive connect indications, or be used to initiate a connection.

```
[ symbol ] TUNBIND [ EP = endpoint_id ]
                   [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                                [ ,SYNC | ASYNC ] ) ]
                   [ ,ECB = INTERNAL | event_control_block_addr ]
                   [ ,EXIT = tpl_exit_routine_address ]
                   [ ,MF = ( I | L | G | M | E,[ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TUNBIND macro instruction executes.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

OPTCD = SHORT | LONG | EXTEND

Format attribute of the parameter list associated with this request.

OPTCD=SHORT a different control block identifier used to indicate that a subset of the TPL was generated.

OPTCD=LONG a standard, full-length TPL is generated.

OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters.

Default: SHORT if MF=I or MF operand omitted, LONG otherwise.

OPTCD = SYNC | ASYNC

Synchronization mode to use when executing the TUNBIND macro instruction.

OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction is complete. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API.

OPTCD=ASYNC the request executes in asynchronous mode, and returns control immediately after scheduling the TUNBIND request. The application program is responsible for issuing the TCHECK macro instruction.

Default: SYNC (synchronous mode).

ECB = INTERNAL | *event_control_block_addr*	Location of an Event Control Block (ECB) to be posted by the API when the TUNBIND macro instruction associated with this TPL completes. The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB. The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB. This operand is mutually exclusive with the following EXIT operand. Default: INTERNAL (internal ECB).
EXIT = *tpl_exit_routine_address*	Address of an exit routine to be scheduled when the TUNBIND macro instruction associated with this TPL completes. The TPL exit routine is scheduled only if asynchronous mode was specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB. This operand is mutually exclusive with the previous ECB operand. Default: Not indicated (no TPL exit routine).
MF = (I | L | G | M | E , [*tpl_address*])	Standard, list, generate, modify, or execute form of the TUNBIND macro instruction. The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used. See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters. Default: MF=I (standard, nonreentrant form).

Completion Information

The TUNBIND macro instruction completes normally when the local protocol address is unbound, and the endpoint disabled. The state of the endpoint is changed from disabled (TSDSABLD) or enabled (TSENABLD) to opened (TSOPENED).

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY). The conditional completion code in register zero is always zero (TCOKAY), and the TPL return code field is set accordingly. No other information is returned.

If the TUNBIND macro instruction completes abnormally, the state of the endpoint remains unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field may contain a specific error code identifying a particular error.

Return Codes

The following table lists the symbolic names for the TUNBIND return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY		
	TAENVIRO	TESYSERR TESTOP	TESUBSYS TETERM	TEDRAIN
	TAFORMAT	TEBDFNCD TEBDEXIT	TEBDOPCD	TEBDECB
	TAPROCED	TEAMODE	TESTATE	TEINCMPL
	TATPLERR	TEACTIVE		
TRFATLFC	func. code	The function code loaded into register zero is invalid.		
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.		

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.

Usage Information

The TUNBIND macro instruction is used to disable an endpoint and disassociate the local protocol address that was bound with the TBIND macro instruction. If the endpoint was operating in connection mode, no connection requests can be initiated or received through the endpoint. If the endpoint was operating in connectionless mode, no datagrams can be sent or received through the endpoint.

After a TUNBIND macro instruction executes, new options can be specified, and another local protocol address can be bound to the endpoint.

TUSER

Associate User with Endpoint— The TUSER macro instruction is used to associate a user ID with an endpoint for accounting and authorization purposes. Any SMF records written contain the user ID, and access to privileged resources or facilities is controlled by access privileges associated with the user ID, and obtained from the local security system.

```
[ symbol ] TUSER [ EP = endpoint_id ]
                 [ ,USER = endpoint_userid ]
                 [ ,OPTCD = ( [ SHORT | LONG | EXTEND ]
                              [ ,SYNC | ASYNC ]
                              [ ,TUB | ACEE ]
                              [ ,PLAIN | CIPHER ] ) ]
                 [ ,ECB = INTERNAL | event_control_block_addr ]
                 [ ,EXIT = tpl_exit_routine_address ]
                 [ ,MF = ( I | L | G | M | E,[ tpl_address ] ) ]
```

EP = *endpoint_id*

Endpoint at which the TUSER macro instruction is to be executed.

The value specified must be the endpoint identifier returned by the TOPEN macro instruction when the endpoint was opened. An invalid or corrupted value causes unpredictable results.

Default: Zero (no endpoint specified).

USER =*endpoint_userid*

Associates a user ID with the endpoint for authorization and accounting purposes.

OPTCD=TUB the specified value must be the address of a Transport Endpoint User Block (TUB) containing the user information.

OPTCD=ACEE the specified value must be the address of an Accessor Environment Element (ACEE) obtained from the local security system when the user ID was authenticated.

If this operand is not coded, the application name specified in the APCB is used.

The password contained in the TUB may be plain text or cipher text depending on the OPTCD=PLAIN | CIPHER operand. If cipher text, it is assumed that the password was encrypted using the encryption mechanism supplied by the local security system. The API merely provides the password to the security system in its encrypted form.

The user ID or application name is also supplied to the transport provider. How this information is used is unspecified, and provider-dependent.

Default: Zero (no user ID; use application name for accounting and authorization).

OPTCD = SHORT | LONG | EXTEND	Format attribute of the parameter list associated with this request. OPTCD=SHORT a different control block identifier used to indicate that a subset of the TPL was generated. OPTCD=LONG a standard, full-length TPL is generated. OPTCD=EXTEND an additional suffix to the standard length TPL is generated. The suffix contains ALET address extensions that can be specified by other request parameters. Default: SHORT if MF=I or MF operand omitted, LONG otherwise.
OPTCD = SYNC | ASYNC	Synchronization mode to use when executing the TUSER macro instruction. OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction completes. A TCHECK macro instruction should not be executed since check processing is automatically performed by the API. OPTCD=ASYNC the request executes in asynchronous mode, and returns control immediately after scheduling the TUSER request. The application program is responsible for issuing the TCHECK macro instruction. Default: SYNC (synchronous mode).
OPTCD = TUB | ACEE	Format of user ID information referenced by the USER operand. OPTCD=TUB user ID, group, and password information is provided in a Transport User Block (TUB). OPTCD=ACEE the user information is contained in an Accessor Environment Element (ACEE) obtained from the local security system. Default: TUB (user information provided in TUB).

OPTCD = PLAIN | CIPHER	Indicates whether the password contained in the Transport User Block (TUB) designated with the USER operand was encrypted, or is in its plain text form. OPTCD=PLAIN the password is in plain text. OPTCD=CIPHER the password is encrypted. The API uses this information when requesting user ID and password verification from the local security system. Default: PLAIN (password in plain text).
ECB = INTERNAL | *event_control_block_addr*	Location of an Event Control Block (ECB) to be posted by the API when the TUSER macro instruction associated with this TPL completes. The ECB can be any fullword of storage aligned on a fullword boundary. If ECB=INTERNAL is coded, the TPL field normally used to store the ECB address is used as an internal ECB. The ECB operand should only be coded when asynchronous mode is specified. In synchronous mode, the request is treated as if ECB=INTERNAL was coded, and any value specified with the ECB operand is overwritten by the internal ECB. This operand is mutually exclusive with the following EXIT operand. Default: INTERNAL (internal ECB).
EXIT = *tpl_exit_routine_address*	Address of an exit routine to be scheduled when the TUSER macro instruction associated with this TPL completes. The TPL exit routine is scheduled only if asynchronous mode was specified. In synchronous mode, any address specified with the EXIT operand is overwritten by an internal ECB. This operand is mutually exclusive with the previous ECB operand. Default: Not indicated (no TPL exit routine).

MF = (I | L | G | M | E , [*tpl_address*])

Standard, list, generate, modify, or execute form of the TUSER macro instruction.

The second sublist operand, *tpl_address*, specifies the address of the TPL to use for this request. If no MF operand is specified, the standard form is used.

See List, Generate, Modify, and Execute Forms for valid combinations of the MF subparameters.

Default: MF=I (standard, nonreentrant form).

Completion Information

The TUSER macro instruction completes normally when the designated user has been associated with the endpoint. All subsequent macro instructions issued at the endpoint are executed with the access privileges of the new user. An SMF record may be written to account for resources utilized by the previous user.

Note: Currently, no authorization checking is performed.

On normal return to the application program, the general return code in register 15 is set to zero (TROKAY). The conditional completion code in register zero is always zero (TCOKAY), and the TPL return code field is set accordingly. No other information is returned.

If the TUSER macro instruction completes abnormally, the previous user continues to be associated with the endpoint, and the state of the endpoint is unchanged. The general return code in register 15, and recovery action code in register zero, indicate the nature of the failure. The TPL return code field can also contain a specific error code that identifies a particular error.

Return Codes

The following table lists the symbolic names for the TUSER return codes. The values associated with the symbolic names can be found in the TPL macro expansion.

General Return Code (Register 15)	Recovery Action Code (Register 0)	Conditional or Specific Error Code/Explanation		
TROKAY	TAOKAY	TCOKAY		
TRFAILED	TAENVIRO	TESYSERR TESTOP	TESUBSYS TETERM	TEDRAIN TEUNAUTH
	TAFORMAT	TEBDFNCD TEBDEXIT	TEBDOPCD TEBDUSER	TEBDECB TEBDACEE
	TAPROCED	TEAMODE	TESTATE	TEINCMPL
	TATPLERR	TEACTIVE		
TRFATLFC	func. code	The function code loaded into register zero is invalid.		
TRFATLPL	diag. code	The TPL address is invalid, or the TPL is corrupted.		
TRFATLAM	diag. code	A fatal access method error occurred, most likely due to corrupted data areas maintained within the application program's address space.		
TRFATLAP	diag. code	The APCB associated with the transport user is closed, or is in the process of closing.		

Usage Information

The TUSER macro instruction is used to associate a new user with an endpoint for authorization and accounting purposes. The new user is identified by a Transport User Block (TUB) or an Accessor Environment Element (ACEE). The access privileges of the new user replace those of the user (if any) defined when the endpoint was opened with a TOPEN macro instruction.

If user information is provided with a TUB, the API authenticates the user ID, group name, and password combination using the local security system. Otherwise, the application must authenticate the user, and provide the address of an ACEE created by the security system.

The TUSER macro instruction is provided for multiple-user application programs that implement a logon procedure. Since the logon information must be gotten from the user after a connection is established, the user ID to be associated with the endpoint is not known when the endpoint is created. In this case, the TOPEN macro instruction should associate the endpoint with the used ID of an overhead account, or use the default privileges associated with the application name (see APCB). When the logon procedure has completed, the real user can be associated with the endpoint by issuing a TUSER macro instruction

Chapter 2

DNR Directory Services

This chapter describes DNR (Domain Name Resolver) directory services and how the DNR can be used to augment the use of API macro instructions.

The DNR is a separate application program and can be configured to run in its own address space or in the same address space as the API. The DNR provides directory services to other application programs. These application programs generally use DNR to acquire network transport services for communicating with remote programs in a distributed processing environment. In fact, DNR uses the API to provide the transport between components of a distributed database that comprises a global network directory.

The following topics are covered in this chapter:

- Directory Database—Describes the Domain Name System (DNS) and local configuration data
- Syntactic Rules for Names—Describes the format and handling of locally managed names and simple domain names
- Directory Services Calls—Includes the calls used to access directory information services for the Domain Name Resolver (DNR)

The Domain Name System (DNS)

The directory consists of locally configured information and global information by supporting the Domain Name System (DNS). The DNS defines a distributed database system and provides a more dynamic information retrieval system than a local database search. You can get information from the DNR by invoking the DIRSRV macro instruction or the dirsrv() C function.

The client provides the name of an object known to the DNR. The DNR searches the directory for information associated with that name or attribute and returns the information to the client.

Directory services can be used to build transport protocol addresses by mapping host names and service names into network addresses and TCP or UDP port numbers.

Other Directory Services

Other directory services are available to support application programs that process and distribute electronic mail. These services are currently provided by DNR:

- Given host name, return:
  - Network address
  - CPU and operating system information
  - List of well-known services supported by host
  - List of host names designated as mail routers for the given host
- Given alias name, return official host name
- Given network address, return official host name
- Given network name, return network number
- Given network number, return network name
- Given protocol name and service name, return associated transport protocol address (that is, TCP or UDP port number)
- Given protocol name, return official protocol number
- Given port number, return associated service name
- Given official protocol number, return protocol name

Directory Database

The DNR information base consists of locally configured information and global information. Corresponding to this data, some services access locally configured information only. Others use both a local and globally distributed database, satisfied by DNR's implementation of the Domain Name System (DNS).

Domain Name System

The Domain Name System (DNS) consists of:

- The domain name space
- Resource records that describe the name space
- Name servers
- Resolvers

The domain name space is a tree structured name space meant to provide a mechanism for naming resources in such a way that the host, network, or protocol is transparent. Each node within the name space corresponds to a set of resources. These resources are identified in a defined data format and referred to as Resource Records (RR). Name servers and resolvers are application programs that exchange the resource records that describe the name space.

The hierarchical structure of the domain name space is reflected in the format of a domain name. Each DNS node is represented by a label that is the simple name of the node. A fully-qualified domain name describes a path through the domain name space to a particular node, starting with the top-level (root) node. The name is formed by concatenating the simple names (or labels) of each node in right-to-left sequence, separated by periods (.) starting with the top-level domain.

Thus, a four-level domain name appears in this format:

level-4.level-3.level-2.level-1

Information about the name space is divided into zones and held by the domain's name servers. A name server's function is to hold information about the domain name space and provide answers to resolver's requests. The resolver's function is to extract information from name servers in response to client requests. The DNR implements the resolver portion of the DNS and queries name servers for responses.

Local Configuration Data

Local configuration data enables DNR to provide services not provided by the Domain Name System and provide an alternative to a network-based directory.

Syntactic Rules for Names

Names provided to DNR by the application program must conform to certain syntactic rules. Any name returned to the application program by DNR conforms to the same rules. These rules are defined separately for locally managed names and Internet domain names.

Locally-Managed Names

Locally-managed names consist of those names defined in locally-maintained configuration data sets, and can be one of the following:

- Alias name (for hosts only)
- Network name
- Service name
- Protocol name

Such names are case-insensitive and consist of alphanumeric characters from the EBCDIC character set (a-z, A-Z, 0-9). You can also use the dash character (-) as long as it is embedded within the name (that is, it does not appear at the beginning or end of the name).

Note: Locally managed names must be less than or equal to 40 characters in length.

Simple Domain Names

Simple domain names (domain name labels) are case-insensitive and consist of alphanumeric characters from the EBCDIC character set (a-z, A-Z, 0-9). You can also use the dash character (-) as long as it is embedded within the name (that is, it does not appear at the beginning or end of the name).

- Simple domain names must be less than or equal to 63 characters in length
- Fully-qualified domain names must be less than or equal to 255 characters in length, including the terminating period

The period at the end of a domain name represents the root of the Domain Name System (DNS) and indicates that the name is fully qualified. A domain name that does not terminate with a period is assumed to be partially qualified. In the latter case, DNR constructs fully-qualified names by appending qualifiers from a search list in a predetermined order.

Directory Services Calls

This section includes the calls used to access directory information services for the Domain Name Resolver (DNR).

Each macro instruction description includes the following information:

- The name of the macro instruction
- A brief statement of its function and use
- The assembler format description
- A detailed description of each operand
- A description of completion information returned
- A table of return codes
- General usage information

Note: It is assumed that you are familiar with the API concepts and facilities presented in TCPaccess *Assembler API Concepts*.

DIRSRV

The DIRSRV macro instruction is defined in terms of its general use in returning information from the directory. Specific instances of the macro instruction are defined that correspond to the specific directory services listed in Other Directory Services earlier in this chapter.

```
[ symbol ]  DIRSRV function_code,

                  information_category,

                  search_argument,
                  NABUF = name_address,
                  NALEN = name_length,
                  VABUF = value_address,
                  VALEN = value_length
                  [ ,QNBUF = qualified_name_address ]
                  [ ,QNLEN = qualified_name_length ]
                  [ ,SYSID = MVS_subsystem_id ]
                  [ ,TIME = time_limit ]
                  [ ,SIZE = size_limit ]
                  [ ,OPTCD = ( [ SYNC | ASYNC ]
                        [ ,BLOCK | NOBLOCK ]
                        [ ,LOCAL | GLOBAL ]
                        [ ,COPY | ORIGINAL ] ) ]
                  [ ,ECB = INTERNAL | event_control_block_addr ]
                  [ ,EXIT = exit_routine_address ]
                  [ ,MF = ( I | L | G | M | E ,[ dpl_address ] ) ]
```

function_code

Valid values are:

GET—Requested information is retrieved from the directory and returned to the application program.

PURGE—A previously issued asynchronous DIRSRV request is purged.

information_category—A category of information maintained by DNR. This operand in combination with the search argument determines the specific type of information requested by the application program.

Valid values are:

HOST—Information regarding a specific host consisting of its official name, alias name, and network-layer addresses (that is, Internet addresses).

NETWORK—Information regarding a specific network consisting of its official name and network number.

SERVICE—Information regarding a well-known service consisting of its official name and transport-layer address (that is, TCP or UDP port number).

PROTOCOL—Information regarding a specific protocol consisting of its official name and protocol number.

HOSTSERV—Information regarding a specific host consisting of its network-layer addresses and associated well-known services available at those addresses.

HOSTINFO—Information regarding a specific host consisting of a standardized CPU and operating system name.

ROUTE—Information regarding electronic mail routes consisting of a list of host names willing to act as mail routers for a specific host.

search_argument

Determines how the directory database is searched by indicating the specific type of information provided by the application program and implying the type of information returned by DNR.

Valid values are:

BYNAME—A name supplied by the application program in the storage area defined by the NABUF and NALEN operands. The value associated with the named object is returned in the storage area defined by the VABUF and VALEN operands.

BYVALUE—A value supplied by the application program in the storage area defined by the VABUF and VALEN operands. The name of the object with the specified value is returned in the storage area defined by the NABUF and NALEN operands.

BYALIA—An alias name (or real name) supplied by the application program in the storage area defined by the NABUF and NALEN operands. The real name associated with the alias name is returned in the storage area defined by the VABUF and VALEN operands.

NABUF = *name_address*

Address of a storage area in which the application program places the name of a host, network, service, or protocol, or in which DNR returns such a name.

The type of name contained or returned in the storage area is determined by the values specified for information_ category and search_argument, and consists of a case-insensitive string of EBCDIC characters. Other syntactic rules may apply depending on the type of name.

Default: Zero (no name storage area).

NALEN = *name_length*

Length (in bytes) of the storage area identified by the NABUF operand. If the storage area contains a name provided by the application program, NALEN is the actual length of the name. If the storage area receives a name returned by DNR, NALEN is the maximum length of the storage area.

The length is updated when the request completes, reflecting the actual length of the name returned. If no information is returned, the request completes abnormally and the length remains unchanged.

A length of zero is invalid.

Default: Zero (no name provided or returned).

VABUF = *value_address*

Address of a storage area in which the application program places:

- An Internet address, network number, TCP or UDP port number
- Transport protocol number

in which DNR returns such a value.

The type of value contained or returned in the storage area is determined by the values specified for information_category and search_argument. The format of such values is type-dependent, and the length is indicated by the VALEN operand.

Default: Zero (no value storage area).

VALEN = *value_length*

Length (in bytes) of the storage area identified by the VABUF operand.

If the storage area contains a value provided by the application program, VALEN is the actual length of the value.

If the storage area receives a value returned by DNR, VALEN is the maximum length of the storage area. The length is updated when the request completes reflecting the actual length of the value returned. If no information is returned, the request completes abnormally and the length remains unchanged.

A length of zero is invalid.

Default: Zero (no value provided or returned).

QNBUF = *qualified_name_address*

Address of a storage area in which DNR returns the fully-qualified name associated with a partial name provided by the application program.

The partially-qualified name is contained in the storage area specified by the NABUF and NALEN operands. Rules governing the formation of names and local configuration information maintained by DNR are used to construct fully-qualified names, which are used to search the directory for the desired information.

The value returned depends on the type of search performed:

- If the search argument is BYALIAS, the name returned is the locally-qualified alias used to search for the real name
- Otherwise, the search argument must be BYNAME and the name returned is the locally-qualified name, or if an alias is encountered in the search, the last de-referenced alias name

Default: Zero (no qualified name storage area).

QNLEN = *qualified_name_length*

Length (in bytes) of the storage area identified by the QNBUF operand.

This length is the maximum length of the storage area and is updated when the request completes reflecting the actual length of the fully-qualified name returned. A length of zero indicates that no name is returned.

Note: If QNLEN is non-zero, QNBUF must also be non-zero and point to the start of a valid storage area.

Default: Zero (no qualified name returned).

SYSID = *MVS_subsystem_id*

ID of the MVS subsystem that processes this request. Normally this operand is not required and an installation default is used.

The *MVS_subsystem_id* is an alphanumeric string up to four characters in length.

Note: If more than one DNR subsystem is active on the local system, you must specify the subsystem you want to process this request.

Default: Not indicated (use installation default).

TIME = *time_limit*

Maximum amount of elapsed time (in seconds) to find the requested information.

Since portions of the distributed database are maintained on other systems, queries must sometimes be transmitted to remote destinations. The responses to such queries can contain referrals to other remote systems. Therefore, some types of directory requests may take an arbitrary amount of time to complete.

You can use this operand to limit the amount of time spent searching for specific information. If the directory search is abandoned because the time limit is exceeded, an error code is returned to the application program.

Default: Zero (use MAXTIME limit in DNRCFG00).

SIZE = *size_limit*

Limits the amount of information returned. The ultimate limit is determined by the length of the storage area provided for receipt of the returned values.

Note: For requests that may return lists or an array of values, this operand limits the number of elements returned. A value of zero indicates there is no limit (other than the size of the storage area provided).

Default: Zero (no size limit).

OPTCD = SYNC | ASYNC	Synchronization mode to use when executing this macro instruction. OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction completes. OPTCD=ASYNC the request executes in asynchronous mode, and returns control to the application program immediately after it is accepted by DNR. The application program is notified asynchronously when the request completes. The ECB and EXIT operands determine the form of asynchronous notification. Default: SYNC (synchronous mode).
OPTCD = BLOCK | NOBLOCK	Disposition of a request if the information is not immediately available. OPTCD=BLOCK processing continues in accordance with the selected synchronization mode (OPTCD=SYNC | ASYNC). This option code affects disposition of a request after it is accepted by DNR. OPTCD=NOBLOCK the request completes abnormally and the specific error code is set to DENOBLOK indicating that the request could not be completed immediately. OPTCD=NOBLOCK is generally used in conjunction with OPTCD=SYNC to prevent suspension of the issuing task for an extended period. **Note:** The issuing task may be momentarily suspended to let the DNR address space process the request. If the application program cannot afford to be suspended, even for a very small amount of time, then the request must be executed asynchronously (OPTCD=ASYNC). Default: BLOCK (suspend task indefinitely).
OPTCD= LOCAL | GLOBAL	Scope of the directory search. OPTCD=LOCAL only the locally maintained database is searched. OPTCD=GLOBAL the distributed database is searched, and queries for the requested information are sent to remote destinations as required. If this option is specified, and access to the network is disabled or inoperative, or no distributed nodes are configured, the request proceeds as if OPTCD=LOCAL was specified. Default: GLOBAL (global scope).

OPTCD =
COPY | ORIGINAL

Specifies whether a copy of the information can be used to complete the request or whether original information must be returned.

OPTCD=COPY a local copy of the requested information is used to satisfy the request.

Note: Specifying OPTCD=COPY lets information gotten from a previous request be reused and returned to the application program, thus avoiding time-consuming queries to remote destinations.

OPTCD=ORIGINAL information must be gotten from its original, authoritative source.

Note: Specifying OPTCD=ORIGINAL assures that the most current information is returned.

Default: COPY (use copy of original data).

ECB = INTERNAL |
event_control_block_addr

Location of an Event Control Block (ECB) posted by DNR when the directory request completes. The ECB can be any fullword of storage aligned on a fullword boundary.

The ECB and EXIT operands share the same storage location in the Directory Services Parameter List (DPL), and are mutually exclusive operands.

If asynchronous mode is specified (OPTCD=ASYNC), the ECB-EXIT field of the DPL (DPLECBXR) is used in the following manner:

ECB=*event_control_block_addr* DNR uses the field as the address of an external ECB. The application program is responsible for issuing a WAIT macro instruction and clearing the ECB when posted.

EXIT=*exit_routine_address* DNR uses the field as the address of an exit routine, and schedules the routine as indicated in EXIT=exit routine address below.

ECB=INTERNAL DNR uses the field as an internal ECB. The application program is responsible for issuing a WAIT macro instruction specifying the internal ECB, but need not clear it when posted.

Note: If synchronous mode is specified (OPTCD=SYNC), the parameter list is flagged for processing as if ECB=INTERNAL was specified, and the ECB-EXIT field is used as an internal ECB that is waited on and cleared automatically before returning control to the application program.

Default: INTERNAL (internal ECB).

EXIT=*exit_routine_address*

Address of a routine scheduled for when the request completes. The EXIT and ECB operands share the same storage location in the parameter list and are mutually exclusive.

The completion exit is scheduled only if asynchronous mode is specified by OPTCD=ASYNC.

If synchronous mode was specified, the exit routine is not used. If one is specified with this operand, the address is overwritten with an internal ECB before the request completes.

Default: Not indicated (no exit routine).

MF = (I | L | G | M | E, [*dpl_address*])

Standard (inline), list, generate, modify, or execute form of the DIRSRV macro instruction.

The second sublist operand, *dp_ address*, specifies the address of the parameter list to use for this request. If no MF operand is specified, the standard form is used.

Note: It is recommended you use the list form of the macro to ensure DPL alignment on a full word boundary. If you do not use the list form, make sure you align the storage area used for the remote parameter list on a fullword boundary. This ensures that the internal ECB is aligned on a fullword boundary.

Default: MF=I (inline, nonreentrant).

Completion Information

The DIRSRV macro instruction completes normally (or conditionally) when the requested information is returned in the storage area provided by the application program. The length of the storage area is updated to reflect the actual amount of information returned.

On normal return to the application program, the general return code in register 15 is set to zero (DROKAY), and the conditional completion code is returned in register zero. The DPL return code field is set accordingly. DCALIAS is set if the name referenced in the NABUF and NALEN operands is not the fully-qualified name associated with the returned data. If DCALIAS is set and a storage area was supplied in the QNBUF and QNLEN operands, the DNR returns the fully-qualified name in the QNBUF storage area. If the fully-qualified name could not fit in the storage area, DCOVRFLO is set. If the application program specified a SIZE limit less than the defined number of entries for the given host, or if the entire list of return information will not fit in the storage area provided, the DCMORE conditional completion code is returned.

If the DIRSRV macro instruction completes abnormally, no information is returned in the storage area and the storage area length is unmodified. The general return code in register 15 and the recovery action code indicate the nature of the failure.

- If the general return code is set to DRFAILED, the recovery action code is returned in register zero and the DPL return code contains a specific error code that identifies a particular error
- If the general return code is set to DRFATLPTL, the recovery action code and the error code are both returned in register zero and the DPL is not updated

If the fully-qualified search string did not conform to the syntactic rules described in Syntactic Rules for Names, a DEBDNAME error code is returned. If the fully-qualified search string is a valid host name but the host does not exist, the DNR returns an error code of DENAMERR. If the fully-qualified search string is a valid and existing host but there is no specific data configured to satisfy the request, the DNR returns an error code of DENODATA. The fully-qualified search string is the result of a fully-qualified name given in the NABUF storage area, a local alias lookup, a name formed by appending the DNR search list strings to a partially qualified domain name, or a DNS alias referral.

Return Codes

The following table lists the symbolic return codes for the DIRSRV macro. The values associated with the symbolic names can be found in the DPL macro expansion.

General Return Code	Recovery Action Code	Conditional or Specific Error Code		
DROKAY	DAOKAY	DCOKAY DCOVRFLO	DCMORE DCNAMEIA	DCALIAS DCLOCAL
DRFAILED	DAEXCPTN	DENONAME DETIMOUT DENOCDS DENOBLOK DEVAMODE	DENOVALU DERFAIL DENAMERR DENODATA DEQNMODE	DENOQNAM DENOTFND DEOVRFLO DENAMODE
	DAENVIRO	DESYSERR DENOTACT DEUNAVBL DETERM	DESUBSYS DENOTRDY DERSOURC	DENOTCNF DESTOP DENOTPRB
	DAFORMAT	DEBDOPCD DEBDEXIT	DEBDFNCD DEBDNAME	DEBDXECB DEBDVALU
DRFATLPTL	DAPROCED	DEACTIVE		
	DADPLERR	DEBDTYPE	DEPROTCT	DEPLMODE

GET-HOST-BYNAME

The GET-HOST-BYNAME instance of the DIRSRV macro instruction is used to return a list of Internet addresses for a given host when its name is known. The name provided can be a local alias or a partially- or fully-qualified domain name, and the information returned can be gotten locally from the host-name configuration data set (DNRHST*xx*) or globally from the Internet Domain Name System (DNS).

```
[ symbol ] DIRSRV GET,HOST,BYNAME,
              NABUF = name_address, |
              NALEN = name_length,
              VABUF = value_address,
              VALEN = value_length
              [ ,QNBUF = qualified_name_address ]
              [ ,QNLEN = qualified_name_length ]
              [ ,SYSID = MVS_subsystem_id ]
              [ ,TIME = time_limit ]
              [ ,SIZE = size_limit ]
              [ ,OPTCD = ( [ SYNC | ASYNC ]
                     [ ,BLOCK | NOBLOCK ]
                     [ ,LOCAL | GLOBAL ]
                     [ ,COPY | ORIGINAL ] )]
              [ ,ECB = INTERNAL | event_control_block_addr ]
              [ ,EXIT = exit_routine_address ]
              [ ,MF = ( I | L | G | M | E, [ dpl_address ] ) ]
```

GET,HOST,BYNAME

The DNR searches for and returns a list of Internet addresses associated with the host name or alias provided by the application program.

NABUF = *name_address*

Address of a storage area in which the application program places the name of a host. The name can be a local alias defined in the alias configuration member (DNRALC*xx*), a local host name defined in the host configuration member (DNRHST*xx*), or an Internet domain name or alias name defined by the DNS. Names can be partially or fully qualified. All names must conform to the syntactic rules described in Syntactic Rules for Names.

Default: Zero (no name storage area).

NALEN = *name_length*

Length (in bytes) of the name located in the storage area identified by the NABUF operand.

The maximum length of a fully-qualified domain name is 255 bytes. The maximum length of a local alias name is 40 bytes. A length of zero is invalid.

Default: Zero (no name provided).

VABUF = *value_address*

Address of a storage area in which DNR returns a list of Internet addresses associated with the named host. Each Internet address is four bytes.

The number of addresses returned is the smaller of the total number defined for the host, the length of the storage area divided by the length of an Internet address (four), or the number indicated by the SIZE operand.

Default: Zero (no value storage area).

VALEN = *value_length*

Length (in bytes) of the storage area identified by the VABUF operand. The length is updated when the request completes reflecting the actual amount of information returned.

If the request completes abnormally, no information is returned and the length remains unchanged. The minimum length of the storage area is four bytes. It is not necessary that the storage area length be evenly divisible by the length of an Internet address.

Default: Zero (no value returned).

QNBUF = *qualified_name_address*

Address of a storage area in which DNR returns the fully-qualified name used to search for the requested information. The fully-qualified name is either the result of a local alias lookup, a name formed by appending the DNR search list strings to a partially qualified domain name, or a DNS alias referral.

Default: Zero (no qualified name storage area).

QNLEN = *qualified_name_length*

Length (in bytes) of the storage area identified by the QNBUF operand.

This length is the maximum length of the storage area and is updated when the request completes reflecting the actual length of the fully-qualified name returned.

Note: If QNLEN is non-zero, QNBUF must also be non-zero and point to the start of a valid storage area.

Default: Zero (no qualified name returned).

SYSID = *MVS_subsystem_id*

ID of the MVS subsystem that will process this request. Normally this operand is not required and an installation default is used.

The *MVS_subsystem_id* is as an alphanumeric string up to four characters.

Note: If more than one DNR subsystem is active on the local system, the particular subsystem that will process this request must be indicated.

Default: Not indicated (use installation default)

TIME = *time_limit*

Maximum amount of elapsed time (in seconds) to find the requested information. Since portions of the distributed database are maintained on other systems, queries must sometimes be transmitted to remote destinations. In addition, the responses to such queries can contain referrals to other remote systems. Therefore, some types of directory requests may take an arbitrary amount of time to complete.

You can use this operand to limit the amount of time spent searching for specific information. If the directory search is abandoned because the time limit is exceeded, DETIMOUT error code is returned to the application program.

Note: TIME is only applicable if OPTCD=GLOBAL.

Default: Zero (use MAXTIME limit in DNRCFG00).

SIZE = *size_limit*	Limits on the amount of information returned for requests that return lists or arrays of values. A value of zero indicates there is no limit and the DNR will return all Internet addresses associated with the domain name. If the return information will not fit in its entirety in the storage area provided or if the application program specifies a limit less than the defined number of Internet addresses for the given host, the DCMORE conditional completion code is returned. Default: Zero (no size limit).
OPTCD = SYNC | ASYNC	The synchronization mode to use when executing this macro instruction. OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction completes. OPTCD=ASYNC the request executes in asynchronous mode, and returns control to the application program immediately after it is accepted by DNR. The application program is notified asynchronously when the request completes. The ECB and EXIT operands determine the form of asynchronous notification. Default: SYNC (synchronous mode).
OPTCD = BLOCK | NOBLOCK	Disposition of a request if the information is not immediately available. OPTCD=BLOCK processing continues in accordance with the synchronization mode selected (OPTCD=SYNC | ASYNC). OPTCD=NOBLOCK the request completes abnormally and the specific error code is set to DENOBLOK indicating that the request could not be completed immediately. OPTCD=NOBLOCK is generally used in conjunction with OPTCD=SYNC to prevent suspension of the issuing task for an extended period. OPTCD=BLOCK|NOBLOCK is only applicable if OPTCD=GLOBAL. This option code affects disposition of a request after it is accepted by the DNR.

	Note: The issuing task can be momentarily suspended to let the DNR address space process the request. If the application program cannot afford to be suspended, even for a very small amount of time, then the request must be executed asynchronously (OPTCD=ASYNC). Default: BLOCK (suspend task indefinitely).
OPTCD = LOCAL | GLOBAL	Scope of the directory search. OPTCD=LOCAL only the locally maintained database is searched. OPTCD=GLOBAL the distributed database is searched, and queries for the requested information are sent to remote destinations as required. If specified, and access to the network is disabled or inoperative, or no distributed nodes are configured, the request proceeds as if OPTCD=LOCAL were specified. Default: GLOBAL (global scope).
OPTCD = COPY | ORIGINAL	Specifies whether a copy of the information is used to complete the request or whether original information must be returned. OPTCD=COPY a local copy of the requested information is used to satisfy the request. Specifying OPTCD=COPY lets information gotten as the result of a previous request be reused and returned to the application program, thus avoiding time-consuming queries to remote destinations. OPTCD=ORIGINAL the information must be gotten from its original, authoritative source, ensuring that the most current information is returned. OPTCD=COPY|ORIGINAL is only applicable if OPTCD=GLOBAL. Default: COPY (use copy of original data).

ECB = INTERNAL | *event_control_block_addr*

Location of an event control block (ECB) posted by DNR when the directory request completes. The ECB can be any fullword of storage aligned on a fullword boundary.

The ECB and EXIT operands share the same storage location in the Directory Services Parameter List (DPL), and are mutually exclusive operands. If asynchronous mode is specified (OPTCD=ASYNC), the ECB-EXIT field of the DPL (DPLECBXR) is used in this manner:

ECB=*event_control_block_addr* DNR uses the field as the address of an external ECB. The application program is responsible for issuing a WAIT macro instruction and clearing the ECB when posted. If EXIT=*exit_routine_address* is specified, DNR uses the field as the address of an exit routine, and schedules the routine as indicated in the following EXIT operand text.

ECB=INTERNAL DNR uses the field as an internal ECB. The application program is responsible for issuing a WAIT macro instruction specifying the internal ECB, but need not clear it when posted.

Note: If synchronous mode is specified (OPTCD=SYNC), the parameter list is flagged for processing as if ECB=INTERNAL was specified, and the ECB-EXIT field is used as an internal ECB that is waited on and cleared automatically before returning control to the application program.

Default: INTERNAL (internal ECB).

EXIT = *exit_routine_address*

Address of the next routine scheduled when the request completes. The EXIT and ECB operands share the same storage location in the parameter list and are mutually exclusive.

The completion exit is scheduled only if asynchronous mode is indicated by OPTCD=ASYNC. If synchronous mode was indicated, the exit routine is not used. If one is specified with this operand, the address is overwritten with an internal ECB before the request completes.

Default: Not indicated (no exit routine).

MF = (I | L | G | M | E, [*dpl_address*])

Standard (inline), list, generate, modify, or execute form of the DIRSRV macro instruction.

The second sublist operand, *dpl_address*, specifies the address of the parameter list to use for this request. If no MF operand is specified, the standard form is used.

Note: It is recommended you use the list form of the macro to ensure DPL alignment on a full word boundary. If you do not use the list form, make sure you align the storage area used for the remote parameter list on a fullword boundary. This ensures that the internal ECB is aligned on a fullword boundary.

Default: MF=I (inline, nonreentrant).

Completion Information

The DIRSRV macro instruction completes normally (or conditionally) when the requested information is returned in the storage area provided by the application program. The length of the storage area is updated to reflect the actual amount of information returned.

On normal return to the application program, the general return code in register 15 is set to zero (DROKAY), and the conditional completion code is returned in register zero. The DPL return code field is set accordingly. DCALIAS is set if the name referenced in the NABUF and NALEN operands are not the fully-qualified name associated with the returned data. If DCALIAS is set and a storage area was supplied in the QNBUF and QNLEN operands, the DNR returns the fully-qualified name in the QNBUF storage area. If the fully-qualified name could not fit in the storage area, DCOVRFLO is set. If the application program specified a SIZE limit less than the defined number of entries for the given host, or if the entire list of return information will not fit in the storage area provided, the DCMORE conditional completion code is returned.

If the DIRSRV macro instruction completes abnormally, no information is returned in the storage area and the storage area length is unmodified. The general return code in register 15 and the recovery action code indicate the nature of the failure.

- If the general return code is set to DRFAILED, the recovery action code is returned in register zero and the DPL return code contains a specific error code that identifies a particular error
- If the general return code is set to DRFATLPTL, the recovery action code and the error code are both returned in register zero and the DPL is not updated

If the fully-qualified search string does not conform to the syntactic rules described in Locally-Managed Names, a DEBDNAME error code is returned. If the fully-qualified search string is a valid host name but the host does not exist, the DNR returns an error code of DENAMERR. If the fully-qualified search string is a valid and existing host but there is no specific data configured to satisfy the request, the DNR returns an error code of DENODATA. The fully-qualified search string is the result of a fully-qualified name given in the NABUF storage area, a local alias lookup, a name formed by appending the DNR search list strings to a partially qualified domain name, or a DNS alias referral.

Return Codes

The following table lists the symbolic return codes for the GET-HOST-BYNAME macro. The values associated with the symbolic names can be found in the DPL macro expansion.

General Return Code	Recovery Action Code	Conditional or Specific Error Code		
DROKAY	DAOKAY	DCOKAY DCOVRFLO	DCMORE DCNAMEIA	DCALIAS DCLOCAL
DRFAILED	DAEXCPTN	DENONAME DETIMOUT DENOCDS DENOBLOK DEVAMODE	DENOVALU DERFAIL DENAMERR DENODATA DEQNMODE	DENOQNAM DENOTFND DEOVRFLO DENAMODE
	DAENVIRO	DESYSERR DENOTACT DEUNAVBL DETERM	DESUBSYS DENOTRDY DERSOURC	DENOTCNF DESTOP DENOTPRB
	DAFORMAT	DEBDOPCD DEBDEXIT	DEBDFNCD DEBDNAME	DEBDXECB DEBDVALU
DRFATLPTL	DAPROCED	DEACTIVE		
	DADPLERR	DEBDTYPE	DEPROTCT	DEPLMODE

Usage Information

The GET-HOST-BYNAME instance of the DIRSRV macro is used to return a list of Internet addresses given an Internet domain name.

- If OPTCD=LOCAL, the DNR returns information received from the host configuration member (DNRHST*xx*)
- If OPTCD=GLOBAL, the DNR returns information received from the Domain Name System Address (A) records

The list of returned Internet addresses is sorted according to the networks given in the network preference configuration member (DNRNPC*xx*).

Example

This example shows the use of GET-HOST-BYNAME. The request is to find the list of internet addresses for the host NS.NASA.GOV. The application supplies this DIRSRV information:

NABUF = (address of:)

	S	.	N	A	S	A	.	G	O	V					

NALEN = 11 VABUF = (address of:)

VALEN = 100 QNBUF = (address of:)

QNLEN = 100
SIZE = 0

This information is returned:

NABUF = (address of:)

NALEN = 11
VABUF = (address of:)

0x66	0x10	0x0a	0xc0	0x34	0xc3	0x0a

VALEN = 8
QNBUF = (address of:)

	S	.	N	A	S	A	.	G	O	V	.				

QNLEN = 12
SIZE = 2

Because the search string, 'NS.NASA.GOV' is partially qualified, the search list strings were appended to 'NS.NASA.GOV'. The root (.) is always included in the search list, therefore, the directory was searched for 'NS.NASA.GOV.'. The DNR received a DNS response for 'NS.NASA.GOV.' and returned the information to the application program.

GET-HOST-BYVALUE

The GET-HOST-BYVALUE instance of the DIRSRV macro instruction is used to return a fully-qualified host name for a host when its address is known. The address provided may be an address defined locally or globally known to the Internet Domain Name System (DNS) and the information returned may be returned locally from the host name configuration member (DNRHST*xx*) or globally.

```
s[ symbol ] DIRSRV GET,HOST,BYVALUE,
        NABUF = name_address,
        NALEN = name_length,
        VABUF = value_address,
        VALEN = value_length
        [ ,SYSID = MVS_subsystem_id ]
        [ ,TIME = time_limit ]
        [ ,OPTCD = ( [ SYNC | ASYNC ]
                [ ,BLOCK | NOBLOCK ]
                [ ,LOCAL | GLOBAL ]
                [ ,COPY | ORIGINAL ] ) ]
        [ ,ECB = INTERNAL | event_control_block_addr ]
        [ ,EXIT = exit_routine_address ]
        [ ,MF = ( I | L | G | M | E, [ dpl_address ] ) ]
```

GET,HOST,BYVALUE — DNR searches for and returns a fully-qualified host name associated with the host Internet address provided by the application program.

NABUF = *name_address* — Address of a storage area in which the DNR returns a fully-qualified host name associated with the given Internet address.

Default: Zero (no name storage area).

NALEN = *name_length*

Length (in bytes) of the storage area identified by the NABUF operand.

The length is updated when the request completes, reflecting the actual amount of information returned. If the request completes abnormally, no information is returned and the length remains unchanged.

Default: Zero (no name returned).

VABUF = *value_address*

Address of a storage area in which the application program placed a four-byte Internet address.

Default: Zero (no value storage area).

VALEN = *value_length*

Length (in bytes) of the storage area identified by the VABUF operand. The value of the storage area must be four bytes.

Default: Zero (no value provided).

SYSID = *MVS_subsystem_id*

ID of the MVS subsystem that will process this request. Normally this operand is not required and an installation default is used.

The *MVS_subsystem_id* is an alphanumeric string up to four characters in length.

Note: If more than one DNR subsystem is active on the local system, the particular subsystem that will process this request must be indicated.

Default: Not indicated (use installation default).

TIME = *time_limit*

Maximum amount of elapsed time (in seconds) to find the requested information.

Since portions of the distributed database are maintained on other systems, queries must sometimes be transmitted to remote destinations. The responses to such queries may contain referrals to other remote systems. Therefore, some types of directory requests can take an arbitrary amount of time to complete.

Use this operand to limit the amount of time spent searching for specific information. If the directory search is abandoned because the time limit was exceeded, DETIMOUT error code is returned the application program.

Note: TIME is only applicable if OPTCD=GLOBAL.

Default: Zero (use MAXTIME limit in DNRCFG00).

OPTCD =
SYNC | ASYNC

Synchronization mode to use when executing this macro instruction.

OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction completes.

OPTCD=ASYNC the request executes in asynchronous mode, and returns control to the application program immediately after it is accepted by DNR. The application program is notified asynchronously when the request completes. The ECB and EXIT operands determine the form of asynchronous notification.

Default: SYNC (synchronous mode).

OPTCD =
BLOCK | NOBLOCK

The disposition of a request if the information is not immediately available.

OPTCD=BLOCK processing continues in accordance with the synchronization mode selected (OPTCD=SYNC | ASYNC).

OPTCD=NOBLOCK the request completes abnormally and the specific error code is set to DENOBLOK indicating the request could not be completed immediately.

Note: OPTCD=NOBLOCK is generally used in conjunction with OPTCD=SYNC to prevent suspension of the issuing task for an extended period of time.

The issuing task may be momentarily suspended to let the DNR address space process the request. If the application program cannot afford to be suspended, even for a very small amount of time, then the request must be executed asynchronously (OPTCD=ASYNC).

OPTCD=BLOCK|NOBLOCK is only applicable if OPTCD=GLOBAL. This option code affects disposition of a request after it is accepted by DNR.

Default: BLOCK (suspend task indefinitely).

OPTCD = LOCAL | GLOBAL	Scope of the directory search. OPTCD=LOCAL only the locally-maintained database is searched. OPTCD=GLOBAL the distributed database is searched, and queries for the requested information are sent to remote destinations as required. If this option is specified, and access to the network is disabled or inoperative, or no distributed nodes are configured, the request proceeds as if OPTCD=LOCAL were specified. OPTCD=COPY|ORIGINAL is only applicable if OPTCD=GLOBAL. Default: GLOBAL (global scope).
OPTCD = COPY | ORIGINAL	Specifies whether a copy of the information can be used to complete the request or whether original information must be returned. OPTCD=COPY use a local copy of the requested information to satisfy the request. Specifying OPTCD=COPY lets information returned as the result of a previous request be reused and returned to the application program, thus avoiding time-consuming queries to remote destinations. OPTCD=ORIGINAL get the information from its original, authoritative source. Specifying OPTCD=ORIGINAL ensures that the most current information is returned. Default: COPY (use copy of original data).

ECB = INTERNAL | *event_control_block_addr*

Location of an event control block (ECB) posted by DNR when the directory request completes. The ECB can be any fullword of storage aligned on a fullword boundary.

The ECB and EXIT operands share the same storage location in the Directory Services Parameter List (DPL), and are mutually exclusive operands. If asynchronous mode is specified (OPTCD=ASYNC), the ECB-EXIT field of the DPL (DPLECBXR) is used in this manner.

ECB=*event_control_block_addr* DNR uses the field as the address of an external ECB. The application program is responsible for issuing a WAIT macro instruction and clearing the ECB when posted. If is specified, DNR uses the field as the address of an exit routine, and schedules the routine as indicated in the following EXIT operand text.

ECB=INTERNAL DNR uses the field as an internal ECB. The application program is responsible for issuing a WAIT macro instruction specifying the internal ECB, but need not clear it when posted.

Note: If synchronous mode is specified (OPTCD=SYNC), the parameter list is flagged for processing as if ECB=INTERNAL was specified, and the ECB-EXIT field is used as an internal ECB that is waited on and cleared automatically before returning control to the application program.

Default: INTERNAL (internal ECB).

EXIT = *exit_routine_address*

Address of a routine to schedule when the request completes. The EXIT and ECB operands share the same storage location in the parameter list and are therefore mutually exclusive.

The completion exit is scheduled only if asynchronous mode is specified by OPTCD=ASYNC. If synchronous mode is specified, the exit routine is not used. If one is specified with this operand, the address is overwritten with an internal ECB before the request completes.

Default: Not indicated (no exit routine).

MF = (I | L | G | M | E, [*dpl_address*])

Standard (inline), list, generate, modify, or execute form of the DIRSRV macro instruction.

The second sublist operand, *dpl_address*, specifies the address of the parameter list to use for this request. If no MF operand is specified, the standard form is used.

Note: It is recommended to use the list form of the macro to ensure DPL alignment on a full word boundary. If you do not use the list form, make sure you align the storage area used for the remote parameter list on a fullword boundary. This ensures that the internal ECB is aligned on a fullword boundary.

Default: MF=I (inline, nonreentrant).

Completion Information

The DIRSRV macro instruction completes normally (or conditionally) when the requested information is returned in the storage area provided by the application program. The length of the storage area is updated to reflect the actual amount of information returned.

On normal return to the application program, the general return code in register 15 is set to zero (DROKAY), and the conditional completion code is returned in register zero. The DPL return code field is set accordingly. If the DIRSRV macro instruction completes abnormally, no information is returned in the storage area and the storage area length is unmodified. The general return code in register 15 and the recovery action code indicate the nature of the failure.

- If the general return code is set to DRFAILED, the recovery action code is returned in register zero and the DPL return code contains a specific error code that identifies a particular error
- If the general return code is set to DRFATLPTL, the recovery action code and the error code are both returned in register zero and the DPL is not updated

Return Codes

The following table lists the symbolic return codes for the GET-HOST-BYVALUE macro. The values associated with the symbolic names can be found in the DPL macro expansion.

General Return Code	Recovery Action Code	Conditional or Specific Error Code		
DROKAY	DAOKAY	DCOKAY	DCLOCAL	
DRFAILED	DAEXCPTN	DENONAME DETIMOUT DENOCDS DENODATA DEQNMODE	DENOVALU DERFAIL DEOVRFLO DENAMODE	DENOQNAMDENOTFND DENOBLOK DEVAMODE
	DAENVIRO	DESYSERR DENOTACT DEUNAVBL DETERM	DESUBSYS DENOTRDY DERSOURC	DENOTCNF DESTOP DENOTPRB
	DAFORMAT	DEBDOPCD DEBDEXIT	DEBDFNCD DEBDNAME	DEBDXECB DEBDVALU
DRFATLPTL	DAPROCED	DEACTIVE		
	DADPLERR	DEBDTYPE	DEPROTCT	DEPLMODE

Usage Information

The GET-HOST-BYVALUE instance of the DIRSRV macro is used to get a fully-qualified domain name given an Internet address.

- If OPTCD=LOCAL, the DNR returns information found in the host configuration member (DNRHST*xx*)
- If OPTCD=GLOBAL, the DNR returns information received from Domain Name System Pointer (PTR) records

Example

This example shows the use of GET-HOST-BYVALUE. The request is to find the host name for Internet address 192.52.195.10.

The application supplies this DIRSRV information:

NABUF = (address of:)

NALEN = 100
VABUF = (address of:)

	0x34	0xc3	0x0a					

VALEN = 4

This information is returned:

NABUF = (address of:)

	S	.	N	A	S	A	.	G	O	V	.				

NALEN = 12
VABUF = (address of:)

	0x34	0xc3	0x0a					

VALEN = 4

GET-HOST-BYALIAS

The GET-HOST-BYALIAS instance of the DIRSRV macro instruction is used to get a fully-qualified domain name for a host when its alias is known. The name provided can be a local alias or a partial or fully-qualified domain name alias, and the information returned may be obtained locally from the alias configuration member (DNRALC*xx*) or globally from the Internet Domain Name System (DNS).

```
[ symbol ]DIRSRV GET,HOST,BYALIAS,
        NABUF = name_address,
        NALEN = name_length,
        VABUF = value_address,
        VALEN = value_length
        [ ,QNBUF = qualified_name_address ]
        [ ,QNLEN = qualified_name_length ]
        [ ,SYSID = MVS_subsystem_id ]
        [ ,TIME = time_limit ]
        [ ,OPTCD = ( [ SYNC | ASYNC ]
                [ ,BLOCK | NOBLOCK ]
                [ ,LOCAL | GLOBAL ]
                [ ,COPY | ORIGINAL ] ) ]
        [ ,ECB = INTERNAL | event_control_block_addr ]
        [ ,EXIT = exit_routine_address ]
        [ ,MF = ( I | L | G | M | E, [ dpl_address ] ) ]
```

GET,HOST,BYALIAS — DNR searches for and returns the fully-qualified name associated with the alias provided by the application program.

NABUF = *name_address* — Address of a storage area in which the application program places the alias name of a host.

The name can be a local alias defined in the alias configuration member (DNRALC*xx*) or an Internet domain alias name defined by the DNS. Names can be partially or fully qualified. All names must conform to the syntactic rules described in Locally-Managed Names.

Default: Zero (no name storage area).

NALEN = *name_length* — Length (in bytes) of the alias located in the storage area identified by the NABUF operand.

The maximum length of a fully-qualified domain name is 255 bytes. The maximum length of a local alias name is 40 bytes. A length of zero is invalid.

Default: Zero (no name provided).

VABUF = *value_address*

Address of a storage area in which the DNR returns a fully-qualified name associated with the given alias.

Default: Zero (no value storage area).

VALEN = *value_length*

Length (in bytes) of the storage area identified by the VABUF operand.

The length is updated when the request completes reflecting the actual amount of information returned. If the request completes abnormally, no information is returned and the length remains unchanged.

Default: Zero (no value returned).

QNBUF = *qualified_name_address*

Address of a storage area in which DNR returns the fully-qualified name used to search for the requested information.

The fully-qualified name is:

- The result of a local alias lookup
- A name formed by appending the DNR search list strings to a partially qualified domain name
- DNS alias referral

Default: Zero (no qualified name storage area).

QNLEN = *qualified_name_length*

Length (in bytes) of the storage area identified by the QNBUF operand.

This length is the maximum length of the storage area and is updated when the request completes reflecting the actual length of the fully-qualified name returned.

Note: If QNLEN is non-zero, QNBUF must also be non-zero and point to the start of a valid storage area.

Default: Zero (no qualified name returned).

SYSID = *MVS_subsystem_id*

ID of the MVS subsystem that will process this request. Normally this operand is not required and an installation default is used.

The *MVS_subsystem_id* is an alphanumeric string up to four characters in length.

Note: If more than one DNR subsystem is active on the local system, the particular subsystem that will process this request must be specified.

Default: Not indicated (use installation default).

TIME = *time_limit*

Maximum amount of elapsed time (in seconds) to find the requested information. Since portions of the distributed database are maintained on other systems, queries must sometimes be transmitted to remote destinations. The responses to such queries may contain referrals to other remote systems. Therefore, some types of directory requests may take an arbitrary amount of time to complete.

You can use this operand to limit the amount of time spent searching for specific information. If the directory search is abandoned because the time limit was exceeded, DETIMOUT error code is returned the application program.

Note: TIME is only applicable if OPTCD=GLOBAL.

Default: Zero (use MAXTIME limit in DNRCFG00).

OPTCD = SYNC | ASYNC

Synchronization mode to use when executing this macro instruction.

OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction completes.

OPTCD=ASYNC the request executes in asynchronous mode, and control is returned to the application program immediately after it is accepted by DNR. The application program is notified asynchronously when the request completes. The ECB and EXIT operands determine the form of asynchronous notification.

Default: SYNC (synchronous mode).

OPTCD =
BLOCK | NOBLOCK

Disposition of a request if the information is not immediately available.

OPTCD=BLOCK processing continues in accordance with the synchronization mode selected (OPTCD=SYNC | ASYNC).

OPTCD=NOBLOCK the request completes abnormally and the specific error code is set to DENOBLOK indicating that the request could not be completed immediately. OPTCD=NOBLOCK is generally used in conjunction with OPTCD=SYNC to prevent suspension of the issuing task for an extended period.

Note: The issuing task may be momentarily suspended to let the DNR address space process the request. If the application program cannot afford to be suspended, even for a very small amount of time, then the request must be executed asynchronously (OPTCD=ASYNC).

OPTCD=BLOCK|NOBLOCK is only applicable if OPTCD=GLOBAL. This option code affects disposition of a request after it is accepted by the DNR.

Default: BLOCK (suspend task indefinitely).

OPTCD =
LOCAL | GLOBAL

Scope of the directory search.

OPTCD=LOCAL only the locally-maintained database is searched.

OPTCD=GLOBAL the distributed database is searched, and queries for the requested information are sent to remote destinations as required.

If this option is specified, and access to the network is disabled or inoperative, or no distributed nodes are configured, the request proceeds as if OPTCD=LOCAL was specified.

Default: GLOBAL (global scope).

OPTCD =
COPY | ORIGINAL

Specifies whether a copy of the information can be used to complete the request or whether original information must be returned.

OPTCD=COPY a local copy of the requested information is used to satisfy the request. Specifying OPTCD=COPY lets information gotten as the result of a previous request be reused and returned to the application program, thus avoiding time-consuming queries to remote destinations.

OPTCD=ORIGINAL the information must be gotten from its original, authoritative source. Specifying OPTCD=ORIGINAL assures that the most current information is returned.

Note: OPTCD=COPY|ORIGINAL is only applicable if OPTCD=GLOBAL.

Default: COPY (use copy of original data).

ECB = INTERNAL |
event_control_block_addr

Location of an event control block (ECB) posted by DNR when the directory request completes. The ECB can be any fullword of storage aligned on a fullword boundary.

The ECB and EXIT operands share the same storage location in the Directory Services Parameter List (DPL), and are therefore mutually exclusive operands. If asynchronous mode is specified (OPTCD=ASYNC), the ECB-EXIT field of the DPL (DPLECBXR) is used in this manner:

If the option is:

ECB=*event_control_block_addr* DNR uses the field as the address of an external ECB. The application program is responsible for issuing a WAIT macro instruction and clearing the ECB when posted. If EXIT= *exit_routine_address* is specified, DNR uses the field as the address of an exit routine, and schedules the routine as indicated in the following EXIT operand text.

ECB=INTERNAL DNR uses the field as an internal ECB. The application program is responsible for issuing a WAIT macro instruction specifying the internal ECB, but need not clear it when posted.

Note: If synchronous mode is specified (OPTCD=SYNC), the parameter list is flagged for processing as if ECB=INTERNAL was specified, and the ECB-EXIT field is used as an internal ECB that is waited on and cleared automatically before returning control to the application program.

Default: INTERNAL (internal ECB).

EXIT = *exit_routine_address*

Address of a routine to schedule when the request completes.

The EXIT and ECB operands share the same storage location in the parameter list and are mutually exclusive. The completion exit is scheduled only if asynchronous mode is indicated by OPTCD=ASYNC.

If synchronous mode is indicated, the exit routine is not used. If one is specified with this operand, the address is overwritten with an internal ECB before the request completes.

Default: Not indicated (no exit routine).

MF = (I | L | G | M | E, [*dpl_address*])

Standard (inline), list, generate, modify, or execute form of the DIRSRV macro instruction.

The second sublist operand, *dpl_address*, specifies the address of the parameter list to use for this request. If no MF operand is specified, the standard form is used.

Note: It is recommended that you use the list form of the macro to ensure DPL alignment on a full word boundary. If you do not use the list form, make sure you align the storage area used for the remote parameter list on a fullword boundary. This ensures that the internal ECB is aligned on a fullword boundary.

Default: MF=I (inline, nonreentrant).

Completion Information

The DIRSRV macro instruction completes normally (or conditionally) when the requested information is returned in the storage area provided by the application program. The length of the storage area is updated to reflect the actual amount of information returned.

On normal return to the application program, the general return code in register 15 is set to zero (DROKAY), and the conditional completion code is returned in register zero. The DPL return code field is set accordingly. DCALIAS is set if the name referenced in the NABUF and NALEN operands is not the fully-qualified name associated with the returned data. If DCALIAS is set and a storage area was supplied in the QNBUF and QNLEN operands, the DNR returns the fully-qualified name in the QNBUF storage area. If the fully-qualified name could not fit in the storage area, DCOVRFLO is set.

If the DIRSRV macro instruction completes abnormally, no information is returned in the storage area and the storage area length is unmodified. The general return code in register 15 and the recovery action code indicate the nature of the failure.

- If the general return code is set to DRFAILED, the recovery action code is returned in register zero and the DPL return code contains a specific error code that identifies a particular error
- If the general return code is set to DRATLPTL, the recovery action code and the error code are both returned in register zero and the DPL is not updated

If the fully-qualified search string did not conform to the syntactic rules described in Syntactic Rules for Names, a DEBDNAME error code is returned. If the fully-qualified search string is a valid host name but the host does not exist, the DNR returns an error code of DENAMERR. If the fully-qualified search string is a valid and existing host but there is no specific data configured to satisfy the request (that is, the search string is not an alias but an actual host name) the DNR returns an error code of DENODATA. The fully-qualified search string is the result of a fully-qualified name given in the NABUF storage area, a local alias lookup, a name formed by appending the DNR search list strings to a partially qualified domain name, or a DNS alias referral.

Return Codes

The following table lists the symbolic return codes for the GET-HOST-BYALIAS macro. The values associated with the symbolic names can be found in the DPL macro expansion.

General Return Code	Recovery Action Code	Conditional or Specific Error Code		
DROKAY	DAOKAY	DCOKAY DCLOCAL	DCALIAS	DCOVRFLO
DRFAILED	DAEXCPTN	DENONAME DETIMOUT DENOCDS DENOBLOK DEVAMODE	DENOVALU DERFAIL DENAMERR DENODATA DEQNMODE	DENOQNAMDENOTFND DEOVRFLO DENAMODE
	DAENVIRO	DESYSERR DENOTACT DEUNAVBL DETERM	DESUBSYS DENOTRDY DERSOURC	DENOTCNF DESTOP DENOTPRB
	DAFORMAT	DEBDOPCD DEBDEXIT	DEBDFNCD DEBDNAME	DEBDXECB DEBDVALU

General Return Code	Recovery Action Code	Conditional or Specific Error Code		
DRFATLPTL	DAPROCED	DEACTIVE		
	DADPLERR	DEBDTYPE	DEPROTCT	DEPLMODE

Usage Information

The GET-HOST-BYALIAS instance of the DIRSRV macro is used to get a fully-qualified domain name given an alias.

- If OPTCD=LOCAL, the DNR returns information received from the alias configuration member (DNRALC*xx*)
- If OPTCD=GLOBAL, the DNR returns information received from Domain Name System Canonical (CNAME) records

Example

This example shows the use of GET-HOST-BYALIAS, OPTCD=LOCAL. The request is to find a fully-qualified domain name for the subsystem name ACSS.

The alias configuration member (DNRALC*xx*) contains:

ACSS ZEUS.ACC.COM.

The application supplies this DIRSRV information:

NABUF = (address of:)

C	S	S

NALEN = 4
VABUF = (address of:)

VALEN = 100

This information is returned:

NABUF = (address of:)

C	S	S

NALEN = 4
VABUF = (address of:)

E	U	S	.	A	C	C	.	C	O	M	.

VALEN = 13

GET-NETWORK-BYNAME

The GET-NETWORK-BYNAME instance of the DIRSRV macro instruction is used to return a network number when its name is known. The information returned is obtained locally from the network configuration member (DNRNET*xx*).

[symbol] DIRSRV GET,NETWORK,BYVALUE,
NABUF = *name_address*,
NALEN = *name_length*,
VABUF = *value_address*,
VALEN = *value_length*
[,SYSID = *MVS_subsystem_id*]
[,OPTCD = SYNC | ASYNC]
[,ECB = INTERNAL | *event_control_block_addr*]
[,EXIT = *exit_routine_address*]
[,MF = (I | L | G | M | E, [*dpl_address*])]

GET,NETWORK,
BYNAME

DNR searches for and returns a network number associated with the network name provided by the application program.

NABUF = *name_address*

Address of a storage area in which the application program places the name of a network. All names must conform to the syntactic rules described in Syntactic Rules for Names.

Default: Zero (no name storage area).

NALEN = *name_length*

Length (in bytes) of the name located in the storage area identified by the NABUF operand.

The maximum length of a local network name is 40 bytes. A length of zero is invalid.

Default: Zero (no name provided or returned).

VABUF = *value_address*

Address of a storage area in which DNR returns a network number associated with the named network.

Default: Zero (no value storage area).

VALEN = *value_length*

Length (in bytes) of the storage area identified by the VABUF operand. The length is updated when the request completes reflecting the actual amount of information returned.

If the request completes normally, the length is updated to a value from one to three.

If the request completes abnormally, no information is returned and the length remains unchanged.

Default: Zero (no value returned).

SYSID = *MVS_subsystem_id*

ID of the MVS subsystem that will process this request. Normally this operand is not required and an installation default is used.

The *MVS_subsystem_id* is an alphanumeric string up to four characters in length.

Note: If more than one DNR subsystem is active on the local system, the particular subsystem that will process this request must be specified.

Default: Not indicated (use installation default).

OPTCD =
SYNC | ASYNC

Synchronization mode to use when executing this macro instruction.

OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction completes.

OPTCD=ASYNC the request executes in asynchronous mode, and control is returned to the application program immediately after it is accepted by DNR. The application program is notified asynchronously when the request completes. The ECB and EXIT operands determine the form of asynchronous notification.

Default: SYNC (synchronous mode).

ECB = INTERNAL |
event_control_block_addr

Location of an event control block (ECB) posted by DNR when the directory request completes. The ECB can be any fullword of storage aligned on a fullword boundary.

The ECB and EXIT operands share the same storage location in the Directory Services Parameter List (DPL), and are mutually exclusive operands. If asynchronous mode is specified (OPTCD=ASYNC), the ECB-EXIT field of the DPL (DPLECBXR) is used in this manner:

ECB=*event_control_block_addr* DNR uses the field as the address of an external ECB. The application program is responsible for issuing a WAIT macro instruction and clearing the ECB when posted. If EXIT= *exit_routine_address* is specified, DNR uses the field as the address of an exit routine, and schedules the routine as indicated in the following EXIT operand text.

ECB=INTERNAL DNR uses the field as an internal ECB. The application program is responsible for issuing a WAIT macro instruction specifying the internal ECB, but need not clear it when posted.

Note: If synchronous mode is specified (OPTCD=SYNC), the parameter list is flagged and will be processed as if ECB=INTERNAL was specified, and the ECB-EXIT field is used as an internal ECB that is waited on and cleared automatically before returning control to the application program.

Default: INTERNAL (internal ECB).

EXIT = *exit_routine_address*

Address of a routine to schedule when the request completes. The EXIT and ECB operands share the same storage location in the parameter list and are therefore mutually exclusive.

The completion exit is scheduled only if asynchronous mode was indicated by OPTCD=ASYNC. If synchronous mode was indicated, the exit routine is not used. If one is specified with this operand, the address is overwritten with an internal ECB before the request completes.

Default: Not indicated (no exit routine).

MF = (I | L | G | M | E, [*dpl_address*])

Standard (inline), list, generate, modify, or execute form of the DIRSRV macro instruction.

The second sublist operand, *dpl_address*, specifies the address of the parameter list to use for this request. If no MF operand is specified, the standard form is used.

Note: It is recommended you use the list form of the macro to ensure DPL alignment on a full word boundary. If you do not use the list form, make sure you align the storage area used for the remote parameter list on a fullword boundary. This ensures that the internal ECB is aligned on a fullword boundary.

Default: MF=I (inline, nonreentrant).

Completion Information

The DIRSRV macro instruction completes normally (or conditionally) when the requested information is returned in the storage area provided by the application program. The length of the storage area is updated to reflect the actual amount of information returned.

On normal return to the application program, the general return code in register 15 is set to zero (DROKAY), and the conditional completion code is returned in register zero. The DPL return code field is set accordingly. If the DIRSRV macro instruction completes abnormally, no information is returned in the storage area and the storage area length is unmodified. The general return code in register 15 and the recovery action code indicate the nature of the failure.

- If the general return code is set to DRFAILED, the recovery action code is returned in register zero and the DPL return code contains a specific error code that identifies a particular error
- If the general return code is set to DRFATLPTL, the recovery action code and the error code are both returned in register zero and the DPL is not updated

Return Codes

The following table lists the symbolic return codes for the GET-NETWORK-BYNAME macro. The values associated with the symbolic names can be found in the DPL macro expansion.

General Return Code	Recovery Action Code	Conditional or Specific Error Code		
DROKAY	DAOKAY	DCOKAY		
DRFAILED	DAEXCPTN	DENONAMEDENOCDS DEVAMODE	DENOVALU DEOVRFLO	DENOTFND DENAMODE
	DAENVIRO	DESYSERR DENOTACT DEUNAVBL DETERM	DESUBSYS DENOTRDY DERSOURC	DENOTCNF DESTOP DENOTPRB
	DAFORMAT	DEBDOPCD DEBDEXIT	DEBDFNCD DEBDNAME	DEBDXECB DEBDVALU
DRFATLPTL	DAPROCED	DEACTIVE		
	DADPLERR	DEBDTYPE	DEPROTCT	DEPLMODE

Example

This example shows the use of GET-NETWORK-BYNAME. The request is to find the network number for the name ARPANET.

The application supplies this DIRSRV information:

NABUF = (address of:)

A	R	P	A	N	E	T

NALEN = 7
VABUF = (address of:)

VALEN = 100

This information is returned:

NABUF = (address of:)

A	R	P	A	N	E	T

NALEN = 7
VABUF = (address of:)

10

VALEN = 1

GET-NETWORK-BYVALUE

The GET-NETWORK-BYVALUE instance of the DIRSRV macro instruction is used to return a network name when its value is known. The information returned is obtained locally from the network configuration member (DNRNET*xx*).

[symbol] DIRSRV GET,NETWORK,BYVALUE,
NABUF = *name_address*,
NALEN = *name_length*,
VABUF = *value_address*,
VALEN = *value_length*
[,SYSID = *MVS_subsystem_id*]
[,OPTCD = SYNC | ASYNC]
[,ECB = INTERNAL | *event_control_block_addr*]
[,EXIT = *exit_routine_address*]
[,MF = (I | L | G | M | E, [*dpl_address*])]

GET,NETWORK,BYVALUE

DNR searches for and returns a network name associated with the network number provided by the application program.

NABUF = *name_address*

The address of a storage area in which the DNR returns the name of a network.

Default: Zero (no name storage area).

NALEN = *name_length*

Length (in bytes) of the storage area identified by the NABUF operand.

The length is updated when the request completes reflecting the actual amount of information returned. If the request completes abnormally, no information is returned and the length remains unchanged.

Default: Zero (no value returned).

VABUF = *value_address*

Address of a storage area in which the application program places a network number.

Default: Zero (no value storage area).

VALEN = *value_length*

Length (in bytes) of the network number located in the storage area identified by the VABUF operand.

This number should be one to three (bytes). A length of zero is invalid.

Default: Zero (no value returned).

SYSID = *MVS_subsystem_id*

ID of the MVS subsystem that will process this request. Normally this operand is not required and an installation default is used.

The *MVS_subsystem_id* is an alphanumeric string up to four characters in length.

Note: If more than one DNR subsystem is active on the local system, the particular subsystem that will process this request must be specified.

Default: Not indicated (use installation default).

OPTCD = SYNC | ASYNC

Synchronization mode used when executing this macro instruction.

OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction completes.

OPTCD=ASYNC the request executes in asynchronous mode, and control is returned to the application program immediately after it is accepted by DNR. The application program is notified asynchronously when the request completes.

Note: The ECB and EXIT operands determine the form of asynchronous notification.

Default: SYNC (synchronous mode).

ECB = INTERNAL | *event_control_block_addr*

Location of an event control block (ECB) posted by DNR when the directory request completes. The ECB can be any fullword of storage aligned on a fullword boundary.

The ECB and EXIT operands share the same storage location in the Directory Services Parameter List (DPL), and are therefore mutually exclusive operands.

If asynchronous mode is specified (OPTCD=ASYNC), the ECB-EXIT field of the DPL (DPLECBXR) is used in this manner:

ECB=*event_control_block_addr* DNR uses the field as the address of an external ECB. The application program is responsible for issuing a WAIT macro instruction and clearing the ECB when posted.

If EXIT= *exit_routine_address* is specified, DNR uses the field as the address of an exit routine, and schedules the routine as indicated in the following EXIT operand text.

ECB=INTERNAL DNR uses the field as an internal ECB. The application program is responsible for issuing a WAIT macro instruction specifying the internal ECB, but need not clear it when posted.

Note: If synchronous mode is specified (OPTCD=SYNC), the parameter list is flagged for processing as if ECB=INTERNAL was specified, and the ECB-EXIT field is used as an internal ECB that is waited on and cleared automatically before returning control to the application program.

Default: INTERNAL (internal ECB).

EXIT = *exit_routine_address*

Address of a routine to schedule when the request completes. The EXIT and ECB operands share the same storage location in the parameter list and are therefore mutually exclusive.

The completion exit is scheduled only if asynchronous mode was specified by OPTCD=ASYNC. If synchronous mode was indicated, the exit routine is not used. If one is specified with this operand, the address is overwritten with an internal ECB before the request completes.

Default: Not indicated (no exit routine).

MF = (I | L | G | M | E, [*dpl_address*])

Standard (inline), list, generate, modify, or execute form of the DIRSRV macro instruction.

The second sublist operand, *dpl_address*, specifies the address of the parameter list to use for this request. If no MF operand is specified, the standard form is used.

Note: It is recommended you use the list form of the macro to ensure DPL alignment on a full word boundary. If you do not use the list form, make sure you align the storage area used for the remote parameter list on a fullword boundary. This ensures that the internal ECB is aligned on a fullword boundary.

Default: MF=I (inline, nonreentrant).

Completion Information

The DIRSRV macro instruction completes normally (or conditionally) when the requested information is returned in the storage area provided by the application program. The length of the storage area is updated to reflect the actual amount of information returned.

On normal return to the application program, the general return code in register 15 is set to zero (DROKAY), and the conditional completion code is returned in register zero. The DPL return code field is set accordingly. If the DIRSRV macro instruction completes abnormally, no information is returned in the storage area and the storage area length is unmodified. The general return code in register 15 and the recovery action code indicate the nature of the failure.

- If the general return code is set to DRFAILED, the recovery action code is returned in register zero and the DPL return code contains a specific error code that identifies a particular error
- If the general return code is set to DRFATLPTL, the recovery action code and the error code are both returned in register zero and the DPL is not updated

Return Codes

The following table lists the symbolic return codes for the GET-NETWORK-BYVALUE macro. The values associated with the symbolic names can be found in the DPL macro expansion.

General Return Code	Recovery Action Code	Conditional or Specific Error Code		
DROKAY	DAOKAY	DCOKAY		
DRFAILED	DAEXCPTN	DENONAME DENOCDS DEVAMODE	DENOVALU DEOVRFLO	DENOTFND DENAMODE
	DAENVIRO	DESYSERR DENOTACT DEUNAVBL DETERM	DESUBSYS DENOTRDY DERSOURC	DENOTCNF DESTOP DENOTPRB
	DAFORMAT	DEBDOPCD DEBDEXIT	DEBDFNCD DEBDNAME	DEBDXECB DEBDVALU
DRFATLPTL	DAPROCED	DEACTIVE		
	DADPLERR	DEBDTYPE	DEPROTCT	DEPLMODE

Example

This example shows the use of GET-NETWORK-BYVALUE. The request is to find the network name for 10.

The application supplies this DIRSRV information:

NABUF = (address of:)

NALEN = 100

VABUF = (address of:)

VALEN = 1

This information is returned:

NABUF = (address of:)

R	P	A	N	E	T

NALEN = 7

VABUF = (address of:)

VALEN = 1

GET-PROTOCOL-BYNAME

The GET-PROTOCOL-BYNAME instance of the DIRSRV macro instruction is used to get a protocol number when its name is known. The information returned is obtained locally from the protocol configuration member (DNRPRT*xx*).

```
[ symbol ] DIRSRV GET,PROTOCOL,BYNAME,
          NABUF = name_address,
          NALEN = name_length,
          VABUF = value_address,
          VALEN = value_length
          [ ,SYSID = MVS_subsystem_id ]
          [ ,OPTCD = SYNC | ASYNC ]
          [ ,ECB = INTERNAL | event_control_block_addr ]
          [ ,EXIT = exit_routine_address ]
          [ ,MF = ( I | L | G | M | E, [ dpl_address ] ) ]
```

GET,PROTOCOL, BYNAME
: DNR searches for and returns a protocol number associated with the protocol name provided by the application program.

NABUF = *name_address*
: Address of a storage area in which the application program has placed a protocol name. All names must conform to the syntactic rules described in Syntactic Rules for Names.

  Default: Zero (no name storage area).

NALEN = *name_length*

Length (in bytes) of the name located in the storage area identified by the NABUF operand.

The maximum length of a protocol name is 40 bytes. A length of zero is invalid.

Default: Zero (no name provided).

VABUF = *value_address*

Address of a storage area in which DNR returns a protocol number associated with the named protocol.

Default: Zero (no value storage area).

VALEN = *value_length*

Length (in bytes) of the storage area identified by the VABUF operand. The length is updated when the request completes reflecting the actual amount of information returned.

If the request completes normally, the length is updated to a value from one to three.

If the request completes abnormally, no information is returned and the length remains unchanged.

Default: Zero (no value returned).

SYSID = *MVS_subsystem_id*

ID of the MVS subsystem that will process this request. Normally this operand is not required and an installation default is used.

The *MVS_subsystem_id* is an alphanumeric string up to four characters in length.

Note: If more than one DNR subsystem is active on the local system, the particular subsystem that will process this request must be specified.

Default: Not indicated (use installation default).

OPTCD = SYNC | ASYNC

Synchronization mode used when executing this macro instruction.

OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction completes.

OPTCD=ASYNC the request executes in asynchronous mode, and returns control to the application program immediately after it is accepted by DNR. The application program is notified asynchronously when the request completes. The ECB and EXIT operands determine the form of asynchronous notification.

Default: SYNC (synchronous mode).

ECB = INTERNAL | *event_control_block_addr*

Location of an event control block (ECB) posted by DNR when the directory request completes. The ECB can be any fullword of storage aligned on a fullword boundary.

The ECB and EXIT operands share the same storage location in the Directory Services Parameter List (DPL), and are therefore mutually exclusive operands. If asynchronous mode is specified (OPTCD=ASYNC), the ECB-EXIT field of the DPL (DPLECBXR) is used in this manner:

ECB=*event_control_block_addr* DNR uses the field as the address of an external ECB. The application program is responsible for issuing a WAIT macro instruction and clearing the ECB when posted. If EXIT= *exit_routine_address* is specified, DNR uses the field as the address of an exit routine, and schedules the routine as indicated in the following EXIT operand text.

ECB=INTERNAL DNR uses the field as an internal ECB. The application program is responsible for issuing a WAIT macro instruction specifying the internal ECB, but need not clear it when posted.

Note: If synchronous mode is specified (OPTCD=SYNC), the parameter list is flagged for processing as if ECB=INTERNAL was specified, and the ECB-EXIT field is used as an internal ECB that is waited on and cleared automatically before returning control to the application program.

Default: INTERNAL (internal ECB).

EXIT=*exit_routine_address*

Address of a routine to schedule when the request completes. The EXIT and ECB operands share the same storage location in the parameter list and are therefore mutually exclusive.

The completion exit is scheduled only if asynchronous mode is specified by OPTCD=ASYNC. If synchronous mode is indicated, the exit routine is not used. If one is specified with this operand, the address is overwritten with an internal ECB before the request completes.

Default: Not indicated (no exit routine).

MF = (I | L | G | M | E, [*dpl_address*])

Standard (inline), list, generate, modify, or execute form of the DIRSRV macro instruction. The second sublist operand, *dpl_address*, specifies the address of the parameter list to use for this request. If no MF operand is specified, the standard form is used.

Note: It is recommended you use the list form of the macro to ensure DPL alignment on a full word boundary. If you do not use the list form, make sure you align the storage area used for the remote parameter list on a fullword boundary. This ensures that the internal ECB is aligned on a fullword boundary.

Default: MF=I (inline, nonreentrant).

Completion Information

The DIRSRV macro instruction completes normally (or conditionally) when the requested information is returned in the storage area provided by the application program. The length of the storage area is updated to reflect the actual amount of information returned.

On normal return to the application program, the general return code in register 15 is set to zero (DROKAY), and the conditional completion code is returned in register zero. The DPL return code field is set accordingly. If the DIRSRV macro instruction completes abnormally, no information is returned in the storage area and the storage area length is unmodified. The general return code in register 15 and the recovery action code indicate the nature of the failure.

- If the general return code is set to DRFAILED, the recovery action code is returned in register zero and the DPL return code contains a specific error code that identifies a particular error
- If the general return code is set to DRFATLPTL, the recovery action code and the error code are both returned in register zero and the DPL is not updated

Return Codes

The following table lists the symbolic return codes for the GET-PROTOCOL-BYNAME macro. The values associated with the symbolic names can be found in the DPL macro expansion.

General Return Code	Recovery Action Code	Conditional or Specific Error Code		
DROKAY	DAOKAY	DCOKAY		
DRFAILED	DAEXCPTN	DENONAMEDENOCDS DEVAMODE	DENOVALU DEOVRFLO	DENOTFND DENAMODE
	DAENVIRO	DESYSERR DENOTACT DEUNAVBL DETERM	DESUBSYS DENOTRDY DERSOURC	DENOTCNF DESTOP DENOTPRB
	DAFORMAT	DEBDOPCD DEBDEXIT	DEBDFNCD DEBDNAME	DEBDXECB DEBDVALU
DRFATLPTL	DAPROCED	DEACTIVE		
	DADPLERR	DEBDTYPE	DEPROTCT	DEPLMODE

Example

This example shows the use of GET-PROTOCOL-BYNAME. The request is to find the protocol number of TCP.

The application supplies this DIRSRV information:

NABUF = (address of:)

C	P

NALEN = 3
VABUF = (address of:)

VALEN = 100

This information is returned:

NABUF = (address of:)

C	P

NALEN = 3
VABUF = (address of:)

VALEN = 1

GET-PROTOCOL-BYVALUE

The GET-PROTOCOL-BYVALUE instance of the DIRSRV macro instruction is used to return a protocol name when its official protocol number is known. The information returned is obtained locally from the protocol configuration member (DNRPRT*xx*).

[symbol] DIRSRV GET,PROTOCOL,BYVALUE,
NABUF = *name_address*,
NALEN = *name_length*,
VABUF = *value_address*,
VALEN = *value_length*
[,SYSID = *MVS_subsystem_id*]
[,SIZE = *size_limit*]
[,ECB = INTERNAL | *event_control_block_addr*]
[,EXIT = *exit_routine_address*]
[,MF = (I | L | G | M | E, [*dpl_address*])]

GET,PROTOCOL,BYVALUE

DNR searches for and returns a protocol name associated with the protocol number provided by the application program.

NABUF = *name_address*

Address of a storage area in which the DNR returns a protocol name associated with the protocol number contained in the storage area indicated by the VABUF and VALEN operands.

Default: Zero (no name storage area).

NALEN = *name_length*

Length (in bytes) of the storage area identified by the NABUF operand.

The length is updated when the request completes reflecting the actual amount of information returned. If the request completes abnormally, no information is returned and the length remains unchanged. A length of zero is invalid.

Default: Zero (no name provided or returned).

VABUF = *value_address*

Address of a storage area in which the application program has placed a protocol number.

Default: Zero (no value storage area).

VALEN = *value_length*

Length (in bytes) of the protocol number located in the storage area identified by the VABUF operand.

This number should be one (byte). A length of zero is invalid.

Default: Zero (no value returned).

SYSID = *MVS_subsystem_id*

ID of the MVS subsystem that will process this request. Normally this operand is not required and an installation default is used.

The *MVS_subsystem_id* is an alphanumeric string up to four characters in length.

Note: If more than one DNR subsystem is active on the local system, the particular subsystem that will process this request must be specified.

Default: Not indicated (use installation default).

SIZE = *size_limit*

Limit on the amount of information returned. The ultimate limit is determined by the length of the storage area provided to receive the returned values.

However, for those requests that may return lists or an array of values, this operand limits the number of elements returned. A value of zero indicates there is no limit (other than the size of the storage area provided).

Default: Zero (no size limit).

ECB = INTERNAL | *event_control_block_addr*

Location of an event control block (ECB) posted by DNR when the directory request completes. The ECB can be any fullword of storage aligned on a fullword boundary.

The ECB and EXIT operands share the same storage location in the Directory Services Parameter List (DPL), and are therefore mutually exclusive operands. If asynchronous mode is specified (OPTCD=ASYNC), the ECB-EXIT field of the DPL (DPLECBXR) is used in this manner:

ECB=*event_control_block_addr* DNR uses the field as the address of an external ECB. The application program is responsible for issuing a WAIT macro instruction and clearing the ECB when posted. If EXIT=*exit_routine_address* is specified, DNR uses the field as the address of an exit routine, and schedules the routine as indicated in the following EXIT operand.

ECB=INTERNAL DNR uses the field as an internal ECB. The application program is responsible for issuing a WAIT macro instruction specifying the internal ECB, but need not clear it when posted.

Note: If synchronous mode is specified (OPTCD=SYNC), the parameter list is flagged for processing as if ECB=INTERNAL was specified, and the ECB-EXIT field is used as an internal ECB that is waited on and cleared automatically before returning control to the application program.

Default: INTERNAL (internal ECB).

EXIT=*exit_routine_address*

Address of a routine to schedule when the request completes.

The EXIT and ECB operands share the same storage location in the parameter list and are therefore mutually exclusive.

The completion exit is scheduled only if asynchronous mode was specified by OPTCD=ASYNC. If synchronous mode is specified, the exit routine is not used. If one is specified with this operand, the address is overwritten with an internal ECB before the request completes.

Default: Not indicated (no exit routine).

MF = (I | L | G | M | E, [*dpl_address*])

Standard (inline), list, generate, modify, or execute form of the DIRSRV macro instruction.

The second sublist operand, *dpl_address*, specifies the address of the parameter list to use for this request. If no MF operand is specified, the standard form is used.

Note: It is recommended you use the list form of the macro to ensure DPL alignment on a full word boundary. If you do not use the list form, make sure you align the storage area used for the remote parameter list on a fullword boundary. This ensures that the internal ECB is aligned on a fullword boundary.

Default: MF=I (inline, nonreentrant).

Completion Information

The DIRSRV macro instruction completes normally (or conditionally) when the requested information is returned in the storage area provided by the application program. The length of the storage area is updated to reflect the actual amount of information returned.

On normal return to the application program, the general return code in register 15 is set to zero (DROKAY), and the conditional completion code is returned in register zero. The DPL return code field is set accordingly. If the DIRSRV macro instruction completes abnormally, no information is returned in the storage area and the storage area length is unmodified. The general return code in register 15 and the recovery action code indicate the nature of the failure.

- If the general return code is set to DRFAILED, the recovery action code is returned in register zero and the DPL return code contains a specific error code that identifies a particular error
- If the general return code is set to DRFATLPTL, the recovery action code and the error code are both returned in register zero and the DPL is not updated

Return Codes

The following table lists the symbolic return codes for the GET-PROTOCOL-BYVALUE macro. The values associated with the symbolic names can be found in the DPL macro expansion.

General Return Code	Recovery Action Code	Conditional or Specific Error Code		
DROKAY	DAOKAY	DCOKAY		
DRFAILED	DAEXCPTN	DENONAME DENOCDS DEVAMODE	DENOVALU DEOVRFLO	DENOTFND DENAMODE
	DAENVIRO	DESYSERR DENOTACT DEUNAVBL DETERM	DESUBSYS DENOTRDY DERSOURC	DENOTCNF DESTOP DENOTPRB
	DAFORMAT	DEBDOPCD DEBDEXIT	DEBDFNCD DEBDNAME	DEBDXECB DEBDVALU
DRFATLPTL	DAPROCED	DEACTIVE		
	DADPLERR	DEBDTYPE	DEPROTCT	DEPLMODE

Example

This example shows the use of GET-PROTOCOL-BYVALUE. The request is to find a protocol name for the protocol number six. The application supplies this DIRSRV information:

NABUF = (address of:)

NALEN = 100
VABUF = (address of:)

VALEN = 1

This information is returned:

NABUF = (address of:)

C	P

NALEN = 3
VABUF = (address of:)

VALEN = 1

GET-SERVICE-BYNAME

The GET-SERVICE-BYNAME instance of the DIRSRV macro instruction is used to return a port number when its name is known. The information returned is obtained locally from the services configuration member (DNRSVC*xx*).

```
[ symbol ] DIRSRV GET,SERVICE,BYNAME,
              NABUF = name_address,
              NALEN = name_length,
              VABUF = value_address,
              VALEN = value_length
              [ ,SYSID = MVS_subsystem_id ]
              [ ,ECB = INTERNAL | event_control_block_addr ]
              [ ,EXIT = exit_routine_address ]
              [ ,MF = ( I | L | G | M | E, [ dpl_address ] ) ]
```

GET,SERVICE,BYNAME — DNR searches for and returns a service port number associated with the protocol/service pair provided by the application program.

NABUF = *name_address* — Address of a storage area in which the application program placed the protocol/service name pairs.

The protocol/service names should be separated by a slash (/). All names must conform to the syntactic rules described in Syntactic Rules for Names.

Default: Zero (no name storage area).

NALEN = *name_length*

Length (in bytes) of the name located in the storage area identified by the NABUF operand.

The maximum length of a local protocol or service name is 40 bytes. A length of zero is invalid.

Default: Zero (no name provided or returned).

VABUF = *value_address*

Address of a storage area in which DNR returns a service number associated with the named protocol/service pair.

Default: Zero (no value storage area).

VALEN = *value_length*

Length (in bytes) of the storage area identified by the VABUF operand. The length is updated when the request completes reflecting the actual amount of information returned.

If the request completes normally, the length is updated to a value from one to three.

If the request completes abnormally, no information is returned and the length remains unchanged.

Default: Zero (no value returned).

SYSID = *MVS_subsystem_id*

ID of the MVS subsystem that will process this request. Normally this operand is not required and an installation default is used.

The *MVS_subsystem_id* is an alphanumeric string up to four characters in length.

Note: If more than one DNR subsystem is active on the local system, the particular subsystem that will process this request must be specified.

Default: Not indicated (use installation default).

ECB = INTERNAL | *event_control_block_addr*

Location of an event control block (ECB) posted by DNR when the directory request completes. The ECB can be any fullword of storage aligned on a fullword boundary.

The ECB and EXIT operands share the same storage location in the Directory Services Parameter List (DPL), and are therefore mutually exclusive operands. If asynchronous mode is specified (OPTCD=ASYNC), the ECB-EXIT field of the DPL (DPLECBXR) is used in this manner:

ECB=*event_control_block_addr* DNR uses the field as the address of an external ECB. The application program is responsible for issuing a WAIT macro instruction and clearing the ECB when posted. If is specified, DNR uses the field as the address of an exit routine, and schedules the routine as indicated in the following EXIT operand.

ECB=INTERNAL DNR uses the field as an internal ECB. The application program is responsible for issuing a WAIT macro instruction specifying the internal ECB, but need not clear it when posted.

Note: If synchronous mode is specified (OPTCD=SYNC), the parameter list is flagged for processing as if ECB=INTERNAL was specified, and the ECB-EXIT field is used as an internal ECB that is waited on and cleared automatically before returning control to the application program.

Default: INTERNAL (internal ECB).

EXIT=*exit_routine_address*

Address of a routine to schedule when the request completes.

The EXIT and ECB operands share the same storage location in the parameter list and are therefore mutually exclusive. The completion exit is scheduled only if asynchronous mode was specified by OPTCD=ASYNC.

If synchronous mode is specified, the exit routine is not used. If one is specified with this operand, the address is overwritten with an internal ECB before the request completes.

Default: Not indicated (no exit routine).

MF = (I | L | G | M | E, [*dpl_address*])

Standard (inline), list, generate, modify, or execute form of the DIRSRV macro instruction.

The second sublist operand, *dpl_address*, specifies the address of the parameter list to use for this request. If no MF operand is specified, the standard form is used.

Note: It is recommended you use the list form of the macro to ensure DPL alignment on a full word boundary. If you do not use the list form, make sure you align the storage area used for the remote parameter list on a fullword boundary. This ensures that the internal ECB is aligned on a fullword boundary.

Default: MF=I (inline, nonreentrant).

Completion Information

The DIRSRV macro instruction completes normally (or conditionally) when the requested information is returned in the storage area provided by the application program. The length of the storage area is updated to reflect the actual amount of information returned.

On normal return to the application program, the general return code in register 15 is set to zero (DROKAY), and the conditional completion code is returned in register zero. The DPL return code field is set accordingly. If the DIRSRV macro instruction completes abnormally, no information is returned in the storage area and the storage area length is unmodified. The general return code in register 15 and the recovery action code indicate the nature of the failure

- If the general return code is set to DRFAILED, the recovery action code is returned in register zero and the DPL return code contains a specific error code that identifies a particular error
- If the general return code is set to DRFATLPTL, the recovery action code and the error code are both returned in register zero and the DPL is not updated

Return Codes

The following table lists the symbolic return codes for the GET-SERVICE-BYNAME macro. The values associated with the symbolic names can be found in the DPL macro expansion.

General Return Code	Recovery Action Code	Conditional or Specific Error Code		
DROKAY	DAOKAY	DCOKAY		
DRFAILED	DAEXCPTN	DENONAME DENOCDS DEVAMODE	DENOVALU DEOVRFLO	DENOTFND DENAMODE
	DAENVIRO	DESYSERR DENOTACT DEUNAVBL DETERM	DESUBSYS DENOTRDY DERSOURC	DENOTCNF DESTOP DENOTPRB
	DAFORMAT	DEBDOPCD DEBDEXIT	DEBDFNCD DEBDNAME	DEBDXECB DEBDVALU
DRFATLPTL	DAPROCED	DEACTIVE		
	DADPLERR	DEBDTYPE	DEPROTCT	DEPLMODE

Example

This example shows the use of GET-SERVICE-BYNAME. The request is to return a service number for the name TCP/DISCARD.

The application supplies this DIRSRV information:

NABUF = (address of:)

C	P	/	D	I	S	C	A	R	D

NALEN = 11
VABUF = (address of:)

VALEN = 100

This information is returned:

NABUF = (address of:)

C	P	/	D	I	S	C	A	R	D

NALEN = 11
VABUF = (address of:)

VALEN = 1

GET-SERVICE-BYVALUE

The GET-SERVICE-BYVALUE instance of the DIRSRV macro instruction is used to return a protocol and service name pair when a service number is known. The information returned is obtained locally from the service configuration member (DNRSRV*xx*).

[*symbol*] DIRSRV GET,SERVICE,BYVALUE,
NABUF = *name_address*,
NALEN = *name_length*,
VABUF = *value_address*,
VALEN = *value_length*
[,SYSID = *MVS_subsystem_id*]
[,SIZE = *size_limit*]
[,ECB = INTERNAL | *event_control_block_addr*]
[,EXIT = *exit_routine_address*_]
[,MF = (I | L | G | M | E, [*dpl_address*])]

GET,SERVICE,
BYVALUE

DNR searches for and returns a list of protocol and service name pairs associated with the service port number provided by the application program.

NABUF = *name_address*

Address of a storage area in which the DNR returns a list of protocol and service name pairs.

The protocol and service name entries are separated by a slash (/). Each entry in the list is separated by a space character. The number of entries returned is returned in the SIZE operand.

Default: Zero (no name storage area).

NALEN = *name_length*

Length (in bytes) of the storage area identified by the NABUF operand.

The length is updated when the request completes reflecting the actual amount of information returned. If the request completes abnormally, no information is returned and the length remains unchanged.

Default: Zero (no value returned).

VABUF = *value_address*

Address of a storage area in which the application program has placed a service port number.

Default: Zero (no value storage area).

VALEN = *value_length*

Length (in bytes) of the port number located in the storage area identified by the VABUF operand.

This number should be two (bytes). A length of zero is invalid.

Default: Zero (no value returned).

SYSID = *MVS_subsystem_id*

ID of the MVS subsystem that will process this request. Normally this operand is not required and an installation default is used.

The *MVS_subsystem_id* is an alphanumeric string up to four characters in length.

Note: If more than one DNR subsystem is active on the local system, the particular subsystem that will process this request must be specified.

Default: Not indicated (use installation default).

SIZE = *size_limit*

Limit of the number of protocol and service name pairs returned.

A value of zero in the request, indicates there is no limit and the DNR is to return all name pairs associated with the port number. If the return information will not fit in the storage area provided or if the application program specifies a limit less than the defined number of protocol and service name pairs, the DCMORE conditional completion code is returned.

Default: Zero (no size limit).

ECB = INTERNAL | *event_control_block_addr*

Location of an event control block (ECB) posted by DNR when the directory request completes. The ECB can be any fullword of storage aligned on a fullword boundary.

The ECB and EXIT operands share the same storage location in the Directory Services Parameter List (DPL), and are therefore mutually exclusive operands. If asynchronous mode is specified (OPTCD=ASYNC), the ECB-EXIT field of the DPL (DPLECBXR) is used in this manner:

ECB=*event_control_block_addr* DNR uses the field as the address of an external ECB. The application program is responsible for issuing a WAIT macro instruction and clearing the ECB when posted. If EXIT=*exit_routine_address* is specified, DNR uses the field as the address of an exit routine, and schedules the routine as indicated in the following EXIT operand text.

ECB=INTERNAL DNR uses the field as an internal ECB. The application program is responsible for issuing a WAIT macro instruction specifying the internal ECB, but need not clear it when posted.

Note: If synchronous mode is specified (OPTCD=SYNC), the parameter list is flagged to be processed as if ECB=INTERNAL had been specified, and the ECB-EXIT field is used as an internal ECB that is waited on and cleared automatically before returning control to the application program.

Default: INTERNAL (internal ECB).

EXIT= *exit_routine_address*

Address of a routine to schedule when the request completes. The EXIT and ECB operands share the same storage location in the parameter list and are therefore mutually exclusive. The completion exit is scheduled only if asynchronous mode is specified by OPTCD=ASYNC.

If synchronous mode was specified, the exit routine is not used. If one is specified with this operand, the address is overwritten with an internal ECB before the request completes.

Default: Not indicated (no exit routine).

MF = (I | L | G | M | E, [*dpl_address*])

Standard (inline), list, generate, modify, or execute form of the DIRSRV macro instruction.

The second sublist operand, *dpl_address*, specifies the address of the parameter list to use for this request. If no MF operand is specified, the standard form is used.

Note: It is recommended you use the list form of the macro to ensure DPL alignment on a full word boundary. If you do not use the list form, make sure you align the storage area used for the remote parameter list on a fullword boundary. This ensures that the internal ECB is aligned on a fullword boundary.

Default: MF=I (inline, nonreentrant).

Completion Information

The DIRSRV macro instruction completes normally (or conditionally) when the requested information is returned in the storage area provided by the application program. The length of the storage area is updated to reflect the actual amount of information returned.

On normal return to the application program, the general return code in register 15 is set to zero (DROKAY), and the conditional completion code is returned in register zero. The DPL return code field is set accordingly. If the application program specified a SIZE limit less than the defined number of entries for the given host, or if the entire list of return information will not fit in the storage area provided, the DCMORE conditional completion code is returned.

If the DIRSRV macro instruction completes abnormally, no information is returned in the storage area and the storage area length is unmodified. The general return code in register 15 and the recovery action code indicate the nature of the failure.

- If the general return code is set to DRFAILED, the recovery action code is returned in register zero and the DPL return code contains a specific error code that identifies a particular error
- If the general return code is set to DRFATLPTL, the recovery action code and the error code are both returned in register zero and the DPL is not updated

Return Codes

The following table lists the symbolic return codes for the GET-SERVICE-BYVALUE macro. The values associated with the symbolic names can be found in the DPL macro expansion.

General Return Code	Recovery Action Code	Conditional or Specific Error Code		
DROKAY	DAOKAY	DCOKAY		
DRFAILED	DAEXCPTN	DENONAME DENOCDS DEVAMODE	DENOVALU DEOVRFLO	DENOTFND DENAMODE
	DAENVIRO	DESYSERR DENOTACT DEUNAVBL DETERM	DESUBSYS DENOTRDY DERSOURC	DENOTCNF DESTOP DENOTPRB
	DAFORMAT	DEBDOPCD DEBDEXIT	DEBDFNCD DEBDNAME	DEBDXECB DEBDVALU
DRFATLPTL	DAPROCED	DEACTIVE		
	DADPLERR	DEBDTYPE	DEPROTCT	DEPLMODE

Example

This example shows the use of GET-SERVICE-BYVALUE. The request is to find the protocol and service name pair for the service port number 9.

The application supplies this DIRSRV information:

NABUF = (address of:)

NALEN = 100
VABUF = (address of:)

VALEN = 1

This information is returned:

NABUF = (address of:)

C P / D I S C A R D U D P

I S C A R D

NALEN = 23
VABUF = (address of:)

VALEN = 1

GET-HOSTINFO-BYNAME

The GET-HOSTINFO-BYNAME instance of the DIRSRV macro instruction is used to return CPU and operating system information for a given host when its name is known. The name provided may be a local alias or a partial or fully-qualified domain name. The information returned is obtained globally from the Internet Domain Name System (DNS).

```
[ symbol ] DIRSRV GET,HOSTINFO,BYNAME,
                NABUF = name_address,
                NALEN = name_length,
                VABUF = value_address,
                VALEN = value_length
                [ ,QNBUF = qualified_name_address ]
                [ ,QNLEN = qualified_name_length ]
                [ ,SYSID = MVS_subsystem_id ]
                [ ,TIME = time_limit ]
                [ ,OPTCD = ( [ SYNC | ASYNC ]
                        [ ,BLOCK | NOBLOCK ]
                        [ ,COPY | ORIGINAL ] ) ]
                [ ,ECB = INTERNAL | event_control_block_addr ]
                [ ,EXIT = exit_routine_address ]
                [ ,MF = ( I | L | G | M | E, [ dpl_address ] ) ]
```

GET,HOSTINFO,BYNAME	DNR searches for and returns CPU and operating system information associated with the host name or alias provided by the application program.
NABUF = *name_address*	Address of a storage area in which the application program has placed the name of a host. The name may be a local alias defined in the alias configuration member (DNRALC*xx*) or an Internet domain name or alias name defined by the DNS. Internet domain names may be partially or fully-qualified. All names must conform to the syntactic rules described in Syntactic Rules for Names. Default: Zero (no name storage area).
NALEN = *name_length*	Length (in bytes) of the name located in the storage area identified by the NABUF operand. The maximum length of a fully-qualified domain name is 255 bytes. The maximum length of a local alias name is 40 bytes. A length of zero is invalid. Default: Zero (no name provided).
VABUF = *value_address*	Address of a storage area in which DNR returns the CPU and operating system information. The strings are EBCIDIC strings separated by a space character. Default: Zero (no value storage area).
VALEN = *value_length*	Length (in bytes) of the storage area identified by the VABUF operand. The length is updated when the request completes reflecting the actual amount of information returned. If the request completes abnormally, no information is returned and the length remains unchanged. Default: Zero (no value returned).
QNBUF = *qualified_name_address*	Address of a storage area in which DNR returns the fully-qualified name used to search for the requested information. The fully-qualified name is either the result of a local alias lookup, a name formed by appending the DNR search list strings to a partially qualified domain name, or a DNS alias referral. Default: Zero (no qualified name storage area).

QNLEN = *qualified_name_length*

Length (in bytes) of the storage area identified by the QNBUF operand.

This length is the maximum length of the storage area and is updated when the request completes reflecting the actual length of the fully-qualified name returned.

Note: If QNLEN is non-zero, QNBUF must also be non-zero and point to the start of a valid storage area.

Default: Zero (no qualified name returned).

SYSID = *MVS_subsystem_id*

ID of the MVS subsystem that will process this request. Normally this operand is not required and an installation default is used.

The *MVS_subsystem_id* is an alphanumeric string up to four characters in length.

Note: If more than one DNR subsystem is active on the local system, the particular subsystem that will process this request must be specified.

Default: Not indicated (use installation default).

TIME = *time_limit*

Maximum amount of elapsed time (in seconds) to find the requested information.

Since portions of the distributed database are maintained on other systems, queries must sometimes be transmitted to remote destinations. Also, the responses to such queries may contain referrals to other remote systems. Therefore, some types of directory requests may take an arbitrary amount of time to complete.

You can use this operand to limit the amount of time spent searching for specific information. If the directory search is abandoned because the time limit was exceeded, DETIMOUT error code is returned the application program.

Default: Zero (use MAXTIME limit in DNRCFG00).

OPTCD = SYNC | ASYNC

Synchronization mode used when executing this macro instruction.

OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction completes.

OPTCD=ASYNC the request executes in asynchronous mode, and returns control to the application program immediately after it is accepted by DNR. The application program is notified asynchronously when the request completes. The ECB and EXIT operands determine the form of asynchronous notification.

Default: SYNC (synchronous mode).

OPTCD = BLOCK | NOBLOCK

Disposition of a request if the information is not immediately available.

OPTCD=BLOCK processing continues in accordance with the synchronization mode selected (OPTCD=SYNC | ASYNC). This option affects disposition of a request after it is accepted by the DNR.

OPTCD=NOBLOCK the request is completed abnormally and the specific error code is set to DENOBLOK indicating that the request could not be completed immediately. OPTCD=NOBLOCK is generally used in conjunction with OPTCD=SYNC to prevent suspension of the issuing task for an extended period.

Note: The issuing task may be momentarily suspended to let the DNR address space process the request. If the application program cannot afford to be suspended, even for a very small amount of time, then the request must be executed asynchronously (OPTCD=ASYNC).

Default: BLOCK (suspend task indefinitely).

OPTCD = COPY | ORIGINAL

Specifies whether a copy of the information can be used to complete the request or whether original information must be returned.

OPTCD=COPY a local copy of the requested information may be used to satisfy the request. Specifying OPTCD=COPY lets information returned as the result of a previous request be reused and returned to the application program, thus avoiding time-consuming queries to remote destinations.

OPTCD=ORIGINAL the information must be returned from its original, authoritative source. Specifying OPTCD=ORIGINAL assures that the most current information is returned.

Default: COPY (use copy of original data).

ECB = INTERNAL | *event_control_block_addr*

Location of an event control block (ECB) posted by DNR when the directory request completes. The ECB can be any fullword of storage aligned on a fullword boundary.

The ECB and EXIT operands share the same storage location in the Directory Services Parameter List (DPL), and are therefore mutually exclusive operands. If asynchronous mode is specified (OPTCD=ASYNC), the ECB-EXIT field of the DPL (DPLECBXR) is used in this manner:

ECB=*event_control_block_addr* DNR uses the field as the address of an external ECB. The application program is responsible for issuing a WAIT macro instruction and clearing the ECB when posted. If EXIT=*exit_routine_address* is specified, DNR uses the field as the address of an exit routine, and schedules the routine as indicated in the following EXIT operand text.

ECB=INTERNAL DNR uses the field as an internal ECB. The application program is responsible for issuing a WAIT macro instruction specifying the internal ECB, but need not clear it when posted.

Note: If synchronous mode is specified (OPTCD=SYNC), the parameter list is flagged for processing as if ECB=INTERNAL was specified, and the ECB-EXIT field is used as an internal ECB that is waited on and cleared automatically before returning control to the application program.

Default: INTERNAL (internal ECB).

EXIT=*exit_routine_address*

Address of a routine to schedule when the request completes. The EXIT and ECB operands share the same storage location in the parameter list and are therefore mutually exclusive.

The completion exit is scheduled only if asynchronous mode is specified by OPTCD=ASYNC. If synchronous mode was indicated, the exit routine is not used. If one is specified with this operand, the address is overwritten with an internal ECB before the request completes.

Default: Not indicated (no exit routine).

MF = (I | L | G | M | E, [*dpl_address*])

Standard (inline), list, generate, modify, or execute form of the DIRSRV macro instruction.

The second sublist operand, *dpl_address*, specifies the address of the parameter list to use for this request. If no MF operand is specified, the standard form is used.

Note: It is recommended you use the list form of the macro to ensure DPL alignment on a full word boundary. If you do not use the list form, make sure you align the storage area used for the remote parameter list on a fullword boundary. This ensures that the internal ECB is aligned on a fullword boundary.

Default: MF=I (inline, nonreentrant).

Completion Information

The DIRSRV macro instruction completes normally (or conditionally) when the requested information is returned in the storage area provided by the application program. The length of the storage area is updated to reflect the actual amount of information returned.

On normal return to the application program, the general return code in register 15 is set to zero (DROKAY), and the conditional completion code is returned in register zero. The DPL return code field is set accordingly. DCALIAS is set if the name referenced in the NABUF and NALEN operands is not the fully-qualified name associated with the returned data. If DCALIAS is set and a storage area was supplied in the QNBUF and QNLEN operands, the DNR returns the fully-qualified name in the QNBUF storage area. If the fully-qualified name could not fit in the storage area, DCOVRFLO is set.

If the DIRSRV macro instruction completes abnormally, no information is returned in the storage area and the storage area length is unmodified. The general return code in register 15 and the recovery action code indicate the nature of the failure.

- If the general return code is set to DRFAILED, the recovery action code is returned in register zero and the DPL return code contains a specific error code that identifies a particular error
- If the general return code is set to DRFATLPTL, the recovery action code and the error code are both returned in register zero and the DPL is not updated

If the fully-qualified search string did not conform to the syntactic rules described in Syntactic Rules for Names, a DEBDNAME error code is returned. If the fully-qualified search string is a valid host name but the host does not exist, the DNR returns an error code of DENAMERR. If the fully-qualified search string is a valid and existing host but there is no specific data configured to satisfy the request, the DNR returns an error code of DENODATA. The fully-qualified search string is the result of a fully-qualified name given in the NABUF storage area, a local alias lookup, a name formed by appending the DNR search list strings to a partially qualified domain name, or a DNS alias referral.

Return Codes

The following table lists the symbolic return codes for the GET-HOSTINFO-BYNAME macro. The values associated with the symbolic names can be found in the DPL macro expansion.

General Return Code	Recovery Action Code	Conditional or Specific Error Code		
DROKAY	DAOKAY	DCOKAY	DCALIAS	DCOVRFLO
DRFAILED	DAEXCPTN	DENONAMEDETIMOUT DENOCDS DENOBLOK DEVAMODE	DENOVALU DERFAIL DENAMERR DENODATA DEQNMODE	DENOQNAMDE NOTFND DEOVRFLO DENAMODE
	DAENVIRO	DESYSERR DENOTACT DEUNAVBL DETERM	DESUBSYS DENOTRDY DERSOURC	DENOTCNF DESTOP DENOTPRB
	DAFORMAT	DEBDOPCD DEBDEXIT	DEBDFNCD DEBDNAME	DEBDXECB DEBDVALU
DRFATLPTL	DAPROCED	DEACTIVE		
	DADPLERR	DEBDTYPE	DEPROTCT	DEPLMODE

Usage Information

The GET-HOSTINFO-BYNAME instance of the DIRSRV macro is used by application programs to return CPU and operating system information. The DNR returns information received from DNS Host Information (HINFO) records. HINFO records specify the data returned by the DNR.

Example

This example shows the use of GET-HOSTINFO-BYNAME. The request is to return CPU and operating system information about UNIX.

The alias configuration member (DNRALC*xx*) contains:

```
ACSS ZEUS.ACC.COM.
UNIX SALT.
```

The application supplies this DIRSRV information:

NABUF = (address of:)

N I X

NALEN = 4
VABUF = (address of:)

VALEN = 100
QNBUF = (address of:)

QNLEN = 100

This information is returned:

NABUF = (address of:)

N I X

NALEN = 4
VABUF = (address of:)

A X - 1 1 / 7 8 5 U N I X

VALEN = 15
QNBUF = (address of

A L T . A C C . C O M .

QNLEN = 13

SALT.ACC.COM. was used to search the directory because UNIX was found as an alias in the alias configuration member. Because the replacement string, SALT, was partially qualified, the search list strings were appended to SALT.

ACC.COM. was included in the search list because the subsystem name ACSS is included in the alias configuration member. The DNR received a DNS response for SALT.ACC.COM. and returned the information to the application program.

GET-HOSTSERV-BYNAME

The GET-HOSTSERV-BYNAME instance of the DIRSRV macro instruction is used to return a list of well-known service supported by a host. The name provided may be a local alias or a partial or fully-qualified domain name. The information returned is obtained globally from the Internet Domain Name System (DNS).

```
[ symbol ] DIRSRV GET,HOSTSERV,BYNAME,
            NABUF = name_address,
            NALEN = name_length,
            VABUF = value_address,
            VALEN = value_length
            [ ,QNBUF=qualified_name_address ]
            [ ,QNLEN=qualified_name_length ]
            [ ,SYSID=MVS_subsystem_id ]
            [ ,TIME=time_limit ]
            [ ,SIZE=size_limit ]
            [ ,OPTCD = ( [ SYNC | ASYNC ]
                    [ ,BLOCK | NOBLOCK ]
                    [ ,COPY | ORIGINAL ] ) ]
            [ ,ECB = INTERNAL | event_control_block_addr ]
            [ ,EXIT = exit_routine_address ]
            [ ,MF = ( I | L | G | M | E, [ dpl_address ] ) ]
```

GET,HOSTSERV,BYNAME	DNR searches for and returns a list of well-known services supported by the host name or alias provided by the application program.
NABUF = *name_address*	Address of a storage area in which the application program has placed the name of a host. The name may be a local alias defined in the alias configuration member (DNRALC*xx*) or an Internet domain name or alias name defined by the DNS. Internet domain names may be partially or fully qualified. All names must conform to the syntactic rules described in Syntactic Rules for Names. Default: Zero (no name storage area).
NALEN = *name_length*	Length (in bytes) of the name located in the storage area identified by the NABUF operand. The maximum length of a fully-qualified domain name is 255 bytes. The maximum length of a local alias name is 40 bytes. A length of zero is invalid. Default: Zero (no name provided).
VABUF = *value_address*	Address of a storage area in which DNR returns a list of well-known services supported by the host. The list contains a four-byte Internet address followed by a two-byte protocol number, a two-byte integer indicating the number of ports supported, followed by an array of two-byte ports. The array size is indicated by the two-byte number proceeding the array. Each entry describes the well-known services supported by a particular protocol on a particular internet address. The number of entries is the smaller of the total number defined for the host, the length of the storage area available, or the number indicated by the SIZE operand. Default: Zero (no value storage area).
VALEN = *value_length*	Length (in bytes) of the storage area identified by the VABUF operand. The length is updated when the request completes reflecting the actual amount of information returned. If the request completes abnormally, no information is returned and the length remains unchanged. Default: Zero (no value returned).

QNBUF = *qualified_name_address*

Address of a storage area in which DNR returns the fully-qualified name used to search for the requested information.

The fully-qualified name is either the result of a local alias lookup, a name formed by appending the DNR search list strings to a partially qualified domain name, or a DNS alias referral.

Default: Zero (no qualified name storage area).

QNLEN = *qualified_name_length*

Length (in bytes) of the storage area identified by the QNBUF operand.

This length is the maximum length of the storage area and is updated when the request completes reflecting the actual length of the fully-qualified name returned.

Note: If QNLEN is non-zero, QNBUF must also be non-zero and point to the start of a valid storage area.

Default: Zero (no qualified name returned).

SYSID = *MVS_subsystem_id*

ID of the MVS subsystem that will process this request. Normally this operand is not required and an installation default is used.

The *MVS_subsystem_id* is an alphanumeric string up to four characters in length.

Note: If more than one DNR subsystem is active on the local system, the particular subsystem that will process this request must be specified.

Default: Not indicated (use installation default).

TIME = *time_limit*

Maximum amount of elapsed time (in seconds) to find the requested information.

Since portions of the distributed database are maintained on other systems, queries must sometimes be transmitted to remote destinations. Also, the responses to such queries may contain referrals to other remote systems. Therefore, some types of directory requests may take an arbitrary amount of time to complete.

You can use this operand to limit the amount of time spent searching for specific information. If the directory search is abandoned because the time limit was exceeded, a DETIMOUT error code is returned to the application program.

Default: Zero (use MAXTIME limit in DNRCFG00).

SIZE = *size_limit*

Limit of the number of entries returned.

A value of zero in the request, indicates there is no limit and the DNR is to return all entries associated with the domain name. If the return information will not fit in the storage area provided or if the application program specifies a limit less than the defined number of entries for the given host, the DCMORE conditional completion code is returned.

Default: Zero (no size limit).

OPTCD = SYNC | ASYNC

Synchronization mode used when executing this macro instruction.

OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction completes.

OPTCD=ASYNC the request executes in asynchronous mode, and returns control to the application program immediately after it is accepted by DNR. The application program is notified asynchronously when the request completes. The ECB and EXIT operands determine the form of asynchronous notification.

Default: SYNC (synchronous mode).

OPTCD = BLOCK | NOBLOCK

Disposition of a request if the information is not immediately available.

OPTCD=BLOCK processing continues in accordance with the synchronization mode selected (OPTCD=SYNC | ASYNC). This option code affects disposition of a request after it is accepted by the DNR.

OPTCD=NOBLOCK the request completes abnormally and the specific error code is set to DENOBLOK indicating that the request could not be completed immediately.

OPTCD=NOBLOCK is generally used in conjunction with OPTCD=SYNC to prevent suspension of the issuing task for an extended period.

Note: The issuing task may be momentarily suspended to let the DNR address space process the request. If the application program cannot afford to be suspended, even for a very small amount of time, then the request must be executed asynchronously (OPTCD=ASYNC).

Default: BLOCK (suspend task indefinitely).

OPTCD = COPY | ORIGINAL

Specifies whether a copy of the information can be used to complete the request or whether original information must be returned.

OPTCD=COPY a local copy of the requested information may be used to satisfy the request. Specifying OPTCD=COPY lets information returned as the result of a previous request be reused and returned to the application program, thus avoiding time-consuming queries to remote destinations.

OPTCD=ORIGINAL the information must be returned from its original, authoritative source. Specifying OPTCD=ORIGINAL assures that the most current information is returned.

Default: COPY (use copy of original data).

ECB = INTERNAL | *event_control_block_addr*

Location of an event control block (ECB) posted by DNR when the directory request completes. The ECB can be any fullword of storage aligned on a fullword boundary.

The ECB and EXIT operands share the same storage location in the Directory Services Parameter List (DPL), and are therefore mutually exclusive operands. If asynchronous mode is specified (OPTCD=ASYNC), the ECB-EXIT field of the DPL (DPLECBXR) is used in this manner:

ECB=*event_control_block_addr* DNR uses the field as the address of an external ECB. The application program is responsible for issuing a WAIT macro instruction and clearing the ECB when posted. If EXIT=*exit_routine_address* is specified, DNR uses the field as the address of an exit routine, and schedules the routine as indicated in the following EXIT operand text.

ECB=INTERNAL DNR uses the field as an internal ECB. The application program is responsible for issuing a WAIT macro instruction specifying the internal ECB, but need not clear it when posted.

Note: If synchronous mode is specified (OPTCD=SYNC), the parameter list is flagged to be processed as if ECB=INTERNAL was specified, and the ECB-EXIT field is used as an internal ECB that is waited on and cleared automatically before returning control to the application program.

Default: INTERNAL (internal ECB).

EXIT=*exit_routine_address*

Address of a routine to schedule when the request completes. The EXIT and ECB operands share the same storage location in the parameter list and are therefore mutually exclusive.

The completion exit is scheduled only if asynchronous mode is specified by OPTCD=ASYNC. If synchronous mode is indicated, the exit routine is not used. If one is specified with this operand, the address is overwritten with an internal ECB before the request completes.

Default: Not indicated (no exit routine).

MF = (I | L | G | M | E, [*dpl_address*])

Standard (inline), list, generate, modify, or execute form of the DIRSRV macro instruction. The second sublist operand, *dpl_address*, specifies the address of the parameter list to use for this request. If no MF operand is specified, the standard form is used.

Note: It is recommended you use the list form of the macro to ensure DPL alignment on a full word boundary. If you do not use the list form, make sure you align the storage area used for the remote parameter list on a fullword boundary. This ensures that the internal ECB is aligned on a fullword boundary.

Default: MF=I (inline, nonreentrant).

Completion Information

The DIRSRV macro instruction completes normally (or conditionally) when the requested information is returned in the storage area provided by the application program. The length of the storage area is updated to reflect the actual amount of information returned.

On normal return to the application program, the general return code in register 15 is set to zero (DROKAY), and the conditional completion code is returned in register zero. The DPL return code field is set accordingly. DCALIAS is set if the name referenced in the NABUF and NALEN operands is not the fully-qualified name associated with the returned data. If DCALIAS is set and a storage area was supplied in the QNBUF and QNLEN operands, the DNR returns the fully-qualified name in the QNBUF storage area. If the fully-qualified name could not fit in the storage area, DCOVRFLO is set. If the application program specified a SIZE limit less than the defined number of entries for the given host, or if the entire list of return information will not fit in the storage area provided, the DCMORE conditional completion code is returned.

If the DIRSRV macro instruction completes abnormally, no information is returned in the storage area and the storage area length is unmodified. The general return code in register 15 and the recovery action code indicate the nature of the failure.

- If the general return code is set to DRFAILED, the recovery action code is returned in register zero and the DPL return code contains a specific error code that identifies a particular error
- If the general return code is set to DRFATLPTL, the recovery action code and the error code are both returned in register zero and the DPL is not updated

If the fully-qualified search string did not conform to the syntactic rules described in Syntactic Rules for Names, a DEBDNAME error code is returned. If the fully-qualified search string is a valid host name but the host does not exist, the DNR returns an error code of DENAMERR. If the fully-qualified search string is a valid and existing host but there is no specific data configured to satisfy the request, the DNR returns an error code of DENODATA. The fully-qualified search string is the result of a fully-qualified name given in the NABUF storage area, a local alias lookup, a name formed by appending the DNR search list strings to a partially qualified domain name, or a DNS alias referral.

Return Codes

The following table lists the symbolic return codes for the GET-HOSTSERV-BYNAME macro. The values associated with the symbolic names can be found in the DPL macro expansion.

General Return Code	Recovery Action Code	Conditional or Specific Error Code		
DROKAY	DAOKAY	DCOKAY DCOVRFLO	DCMORE	DCALIAS
DRFAILED	DAEXCPTN	DENONAME DETIMOUT DENOCDS DENOBLOK DEVAMODE	DENOVALU DERFAIL DENAMERR DENODATA DEQNMODE	DENOQNAM DENOTFND DEOVRFLO DENAMODE
	DAENVIRO	DESYSERR DENOTACT DEUNAVBL DETERM	DESUBSYS DENOTRDY DERSOURC	DENOTCNF DESTOP DENOTPRB
	DAFORMAT	DEBDOPCD DEBDEXIT	DEBDFNCD DEBDNAME	DEBDXECB DEBDVALU
DRFATLPTL	DAPROCED	DEACTIVE		
	DADPLERR	DEBDTYPE	DEPROTCT	DEPLMODE

Usage Information

The GET-HOSTSERV-BYNAME instance of the DIRSRV macro is used by application programs to determine the services supported by a given Internet domain name. The DNR returns information received from DNS Well Known Services (WKS) records. WKS records specify the well-known services supported by a particular protocol on a particular internet address. The two-byte protocol number returned in an entry may be used as input in a DIRSRV GET-PROTOCOL-BYVALUE request to return the associated protocol name. Likewise, the two byte port numbers may be used as input in a DIRSRV GET-SERVICE-BYVALUE request to obtain the associated service names.

Example

This example shows the use of GET-HOSTSERV-BYNAME. The request is to find the list of well-known services supported by host TERP.UMD.EDU.

The application supplies this DIRSRV information:

NABUF = (address of:)

E R P . U M D . E D U

NALEN = 12
VABUF = (address of:)

VALEN = 100
QNBUF = (address of:)

QNLEN = 100
SIZE = 100

This information is returned:

NABUF = (address of:)

E R P . U M D . E D U .

NALEN = 13
VABUF = (address of:)

0x08	0x0a	0x5a	17	2	53

0x08	0x0a	0x5a	6	3	21

25

VALEN = 26
QNBUF = (address of:)

QNLEN = 100
SIZE = 2

The response information indicates that the Internet address 128.8.10.90 (0x80080a5a) supports two UDP (17) well-known services DOMAIN (53) and NTP (123) and 3 TCP (6) services FTP (21), TELNET (23), and SMTP (25).

GET-ROUTE-BYNAME

The GET-ROUTE-BYNAME instance of the DIRSRV macro instruction is used to return a list of hosts willing to act as a mail exchange for a host. The name provided may be a local alias or a partial or fully-qualified domain name. The information returned is obtained globally from the Internet Domain Name System (DNS).

```
[ symbol ] DIRSRV GET,ROUTE,BYNAME,
                NABUF = name_address,
                NALEN = name_length,
                VABUF = value_address,
                VALEN = value_length
                [ ,QNBUF = qualified_name_address ]
                [ ,QNLEN = qualified_name_length ]
                [ ,SYSID = MVS_subsystem_id ]
                [ ,TIME = time_limit ]
                [ ,SIZE = size_limit ]
                [ ,OPTCD = ( [ SYNC | ASYNC ]
                        [ ,BLOCK | NOBLOCK ]
                        [ ,COPY | ORIGINAL ] ) ]
                [ ,ECB = INTERNAL | event_control_block_addr ]
                [ ,EXIT = exit_routine_address ]
                [ ,MF = ( I | L | G | M | E, [ dpl_address ] ) ]
```

GET,ROUTE,BYNAME — DNR searches for and returns a list of host names willing to act as mail exchanges for the host name or alias provided by the application program.

NABUF = *name_address* — Address of a storage area in which the application program has placed the name of a host.

The name may be a local alias defined in the alias configuration member (DNRALC*xx*) or an Internet domain name or alias name defined by the DNS. Internet domain names may be partially or fully-qualified. All names must conform to the syntactic rules described in Syntactic Rules for Names.

Default: Zero (no name storage area).

NALEN = *name_length* — Length (in bytes) of the name located in the storage area identified by the NABUF operand.

The maximum length of a fully-qualified domain name is 255 bytes. The maximum length of a local alias name is 40 bytes. A length of zero is invalid.

Default: Zero (no name provided).

VABUF = *value_address*

Address of a storage area in which DNR returns a list of hosts willing to act as mail exchanges.

The list contains EBCDIC character strings separated by a space character. The number of strings returned is specified in the SIZE operand. The list is sorted by preference giving the highest preferred host first. The number of entries is the smaller of the total number defined for the host, the length of the storage area available, or the number indicated by the SIZE operand.

Default: Zero (no value storage area).

VALEN = *value_length*

Length (in bytes) of the storage area identified by the VABUF operand.

The length is updated when the request completes reflecting the actual amount of information returned. If the request completes abnormally, no information is returned and the length remains unchanged.

Default: Zero (no value returned).

QNBUF = *qualified_name_address*

Address of a storage area in which DNR returns the fully-qualified name used to search for the requested information.

The fully-qualified name is either the result of a local alias lookup, a name formed by appending the DNR search list strings to a partially qualified domain name, or a DNS alias referral.

Default: Zero (no qualified name storage area).

QNLEN = *qualified_name_length*

Length (in bytes) of the storage area identified by the QNBUF operand.

This length is the maximum length of the storage area and is updated when the request completes reflecting the actual length of the fully-qualified name returned.

Note: If QNLEN is non-zero, QNBUF must also be non-zero and point to the start of a valid storage area.

Default: Zero (no qualified name returned).

SYSID = M*VS_subsystem_id*

ID of the MVS subsystem that will process this request. Normally this operand is not required and an installation default is used.

The *MVS_subsystem_id* is an alphanumeric string up to four characters in length.

Note: If more than one DNR subsystem is active on the local system, the particular subsystem that will process this request must be specified.

Default: Not indicated (use installation default).

TIME = *time_limit*

Maximum amount of elapsed time (in seconds) to find the requested information.

Since portions of the distributed database are maintained on other systems, queries must sometimes be transmitted to remote destinations. Also, the responses to such queries may contain referrals to other remote systems. Therefore, some types of directory requests may take an arbitrary amount of time to complete.

You can use this operand to limit the amount of time spent searching for specific information. If the directory search is abandoned because the time limit was exceeded, a DETIMOUT error code is returned to the application program.

Default: Zero (use MAXTIME limit in DNRCFG00).

SIZE = *size_limit*

Limit of the number of entries returned.

A value of zero in the request, indicates there is no limit and the DNR is to return all entries associated with the domain name. If the return information will not fit in the storage area provided or if the application program specifies a limit less than the defined number of entries for the given host, the DCMORE conditional completion code is returned.

Default: Zero (no size limit).

OPTCD =
SYNC | ASYNC

Synchronization mode used when executing this macro instruction.

OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction completes.

OPTCD=ASYNC the request executes in asynchronous mode, and returns control to the application program immediately after it is accepted by DNR. The application program is notified asynchronously when the request completes. The ECB and EXIT operands determine the form of asynchronous notification.

Default: SYNC (synchronous mode).

OPTCD =
BLOCK | NOBLOCK

Disposition of a request if the information is not immediately available.

OPTCD=BLOCK processing continues in accordance with the synchronization mode selected (OPTCD=SYNC | ASYNC). This option code affects disposition of a request after it is accepted by the DNR.

OPTCD=NOBLOCK the request completes abnormally and the specific error code is set to DENOBLOK indicating that the request could not be completed immediately. OPTCD=NOBLOCK is generally used in conjunction with OPTCD=SYNC to prevent suspension of the issuing task for an extended period of time.

Note: The issuing task may be momentarily suspended to let the DNR address space process the request. If the application program cannot afford to be suspended, even for a very small amount of time, then the request must be executed asynchronously (OPTCD=ASYNC).

Default: BLOCK (suspend task indefinitely).

OPTCD =
COPY | ORIGINAL

Specifies whether a copy of the information can be used to complete the request or whether original information must be returned.

OPTCD=COPY a local copy of the requested information can be used to satisfy the request. Specifying OPTCD=COPY lets information returned as the result of a previous request be reused and returned to the application program, thus avoiding time-consuming queries to remote destinations.

OPTCD=ORIGINAL the information must be returned from its original, authoritative source. Specifying OPTCD=ORIGINAL assures that the most current information is returned.

Default: COPY (use copy of original data).

ECB = INTERNAL | *event_control_block_addr*

Location of an event control block (ECB) posted by DNR when the directory request completes. The ECB can be any fullword of storage aligned on a fullword boundary.

The ECB and EXIT operands share the same storage location in the Directory Services Parameter List (DPL), and are therefore mutually exclusive operands. If asynchronous mode is specified (OPTCD=ASYNC), the ECB-EXIT field of the DPL (DPLECBXR) is used in this manner:

ECB=*event_control_block_addr* DNR uses the field as the address of an external ECB. The application program is responsible for issuing a WAIT macro instruction and clearing the ECB when posted. If EXIT=*exit_routine_address* is specified, DNR uses the field as the address of an exit routine, and schedules the routine as indicated in THE SECTION "Syntax Description".

If the option is ECB=INTERNAL, DNR uses the field as an internal ECB. The application program is responsible for issuing a WAIT macro instruction specifying the internal ECB, but need not clear it when posted.

Note: If synchronous mode is specified (OPTCD=SYNC), the parameter list is flagged for processing as if ECB=INTERNAL was specified, and the ECB-EXIT field is used as an internal ECB that is waited on and cleared automatically before returning control to the application program.

Default: INTERNAL (internal ECB).

EXIT=*exit_routine_address*

Address of a routine to schedule when the request completes. The EXIT and ECB operands share the same storage location in the parameter list and are therefore mutually exclusive.

The completion exit is scheduled only if asynchronous mode is specified by OPTCD=ASYNC. If synchronous mode was specified, the exit routine is not used. If one is specified with this operand, the address is overwritten with an internal ECB before the request completes.

Default: Not indicated (no exit routine).

MF = (I | L | G | M | E, [*dpl_address*])

Standard (inline), list, generate, modify, or execute form of the DIRSRV macro instruction.

The second sublist operand, *dpl_address*, specifies the address of the parameter list to use for this request. If no MF operand is specified, the standard form is used.

Note: It is recommended you use the list form of the macro to ensure DPL alignment on a full word boundary. If you do not use the list form, make sure you align the storage area used for the remote parameter list on a fullword boundary. This ensures that the internal ECB is aligned on a fullword boundary.

Default: MF=I (inline, nonreentrant).

Completion Information

The DIRSRV macro instruction completes normally (or conditionally) when the requested information is returned in the storage area provided by the application program. The length of the storage area is updated to reflect the actual amount of information returned.

On normal return to the application program, the general return code in register 15 is set to zero (DROKAY), and the conditional completion code is returned in register zero. The DPL return code field is set accordingly. DCALIAS is set if the name referenced in the NABUF and NALEN operands is not the fully-qualified name associated with the returned data. If DCALIAS is set and a storage area was supplied in the QNBUF and QNLEN operands, the DNR returns the fully-qualified name in the QNBUF storage area. If the fully-qualified name could not fit in the storage area, DCOVRFLO is set. If the application program specified a SIZE limit less than the defined number of entries for the given host, or if the entire list of return information will not fit in the storage area provided, the DCMORE conditional completion code is returned.

If the DIRSRV macro instruction completes abnormally, no information is returned in the storage area and the storage area length is unmodified. The general return code in register 15 and the recovery action code indicate the nature of the failure.

- If the general return code is set to DRFAILED, the recovery action code is returned in register zero and the DPL return code contains a specific error code that identifies a particular error
- If the general return code is set to DRFATLPTL, the recovery action code and the error code are both returned in register zero and the DPL is not updated

If the fully-qualified search string did not conform to the syntactic rules described in Syntactic Rules for Names, a DEBDNAME error code is returned. If the fully-qualified search string is a valid host name but the host does not exist, the DNR returns an error code of DENAMERR. If the fully-qualified search string is a valid and existing host but there is no specific data configured to satisfy the request, the DNR returns an error code of DENODATA. The fully-qualified search string is the result of a fully-qualified name given in the NABUF storage area, a local alias lookup, a name formed by appending the DNR search list strings to a partially qualified domain name, or a DNS alias referral.

Return Codes

The following table lists the symbolic return codes for the GET-ROUTE-BYNAME macro. The values associated with the symbolic names can be found in the DPL macro expansion.

General Return Code	Recovery Action Code	Conditional or Specific Error Code		
DROKAY	DAOKAY	DCOKAY DCOVRFLO	DCMORE DCLOCAL	DCALIAS
DRFAILED	DAEXCPTN	DENONAMEDETIM OUT DENOCDS DENOBLOK DEVAMODE	DENOVALU DERFAIL DENAMERR DENODATA DEQNMODE	DENOQNAM DENOTFND DEOVRFLO DENAMODE
	DAENVIRO	DESYSERR DENOTACT DEUNAVBL DETERM	DESUBSYS DENOTRDY DERSOURC	DENOTCNF DESTOP DENOTPRB
	DAFORMAT	DEBDOPCD DEBDEXIT	DEBDFNCD DEBDNAME	DEBDXECB DEBDVALU
DRFATLPTL	DAPROCED	DEACTIVE		
	DADPLERR	DEBDTYPE	DEPROTCT	DEPLMODE

Usage Information

The GET-ROUTE-BYNAME instance of the DIRSRV macro is used by mailers to determine how to route a message given an Internet domain name. The DNR returns information received from DNS Mail Exchange (MX) records. MX records specify mail exchange hosts that know how to route mail to a host not directly connected to the Internet. MX records include a preference value that specifies the preference given to the host relative to other hosts. The DNR sorts MX records according to preference (lower values are preferred) and returns the host names in order. This order reflects the order the calling mailer programs should follow when delivering mail. To determine the internet address of the remote hosts, mailers should pass the hosts names returned in the GET-ROUTE-BYNAME request as search strings to a DIRSRV GET-HOST-BYNAME request.

Example

This example shows the use of GET-ROUTE-BYNAME. The request is to find a list of hosts willing to act as a mail exchange for the host UNIX. The alias configuration member (DNRALC*xx*) contains:

UNIX VAX.ACC.COM.

The application supplies this DIRSRV information:

NABUF = (address of:)

N	I	X

NALEN = 4
VABUF = (address of:)

VALEN = 100
QNBUF = (address of:)

QNLEN = 100
SIZE = 100

This information is returned:

NABUF = (address of:)

N I X

NALEN = 4
VABUF = (address of:)

A T U R N . A C C . C O M .
A L T . A C C . C O M .

VALEN = 29
QNBUF = (address of:)

A T U R N . A C C . C O M .

QNLEN = 15
SIZE = 2

SATURN.ACC.COM. is returned in the QNBUF because the search string UNIX was listed as an alias in the alias configuration member. The replacement string VAX.ACC.COM. was in turn known as an alias of SATURN.ACC.COM. in the domain name space. The DNR received a DNS response for SATURN.ACC.COM. and returned the information to the application program.

GET-RPC-BYNAME

The GET-RPC-BYNAME instance of the DIRSRV macro instruction is used to return an RPC number when its RPC name is known. This information returned is obtained locally from the RPC configuration member (DNRRPC*xx*).

```
[ symbol ] DIRSRV GET,RPC,BYNAME,
                NABUF = name_address,
                NALEN = name_length,
                VABUF = value_address,
                VALEN = value_length
                [ ,QNBUF = qualified_name_address ]
                [ ,QNLEN = qualified_name_length ]
                [ ,SYSID = MVS_subsystem_id ]
                [ ,OPTCD = SYNC | ASYNC ]
                [ ,ECB = INTERNAL | event_control_block_addr ]
                [ ,EXIT = exit_routine_address ]
                [ ,MF = ( I | L | G | M | E, [ dpl_address ] ) ]
```

GET,RPC,BYNAME

DNR is to search for and return an RPC value associated with the RPC name provided by the application program.

NABUF = *name_address*

Address of a storage area in which the application program has placed an RPC name.

The name may be an RPC alias defined in the RPC configuration member (DNRRPC*xx*) or an official RPC name.

Default: Zero (no name storage area).

NALEN = *name_length*

Length (in bytes) of the name located in the storage area identified by the NABUF operand.

The maximum length of an RPC name is 40 bytes. A length of zero is invalid.

Default: Zero (no name provided).

VABUF = *value_address*

Address of a storage area in which DNR returns an RPC value associated with the RPC name. Each returned RPC value is four bytes.

Default: Zero (no value storage area).

VALEN = *value_length*

Length (in bytes) of the storage area identified by the VABUF operand.

The length is updated when the request completes reflecting the actual amount of information returned. If the request completes abnormally, no information is returned and the length remains unchanged. The minimum length of the storage area is four bytes.

Default: Zero (no value returned).

QNBUF = *qualified_name_address*

Address of a storage area in which DNR returns the official RPC name used to search for the requested information. The DNR returns an RPC name in the QNBUF data area if the search string was an alias.

Default: Zero (no qualified name storage area).

QNLEN = *qualified_name_length*

Length (in bytes) of the storage area identified by the QNBUF operand.

This length is the maximum length of the storage area and is updated when the request completes reflecting the actual length of the fully-qualified name returned.

Note: If QNLEN is non-zero, QNBUF must also be non-zero and point to the start of a valid storage area.

Default: Zero (no qualified name returned).

SYSID = *MVS_subsystem_id*

ID of the MVS subsystem that will process this request. Normally this operand is not required and an installation default is used.

The *MVS_subsystem_id* is an alphanumeric string up to four characters in length.

Note: If more than one DNR subsystem is active on the local system, the particular subsystem that will process this request must be specified.

Default: Not indicated (use installation default).

OPTCD =
SYNC | ASYNC

Synchronization mode used when executing this macro instruction.

OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction completes.

OPTCD=ASYNC the request executes in asynchronous mode, and returns control to the application program immediately after it is accepted by DNR. The application program is notified asynchronously when the request completes. The ECB and EXIT operands determine the form of asynchronous notification.

Default: SYNC (synchronous mode).

ECB = INTERNAL |
event_control_block_addr

Location of an event control block (ECB) posted by DNR when the directory request completes. The ECB can be any fullword of storage aligned on a fullword boundary.

The ECB and EXIT operands share the same storage location in the Directory Services Parameter List (DPL), and are therefore mutually exclusive operands. If asynchronous mode is specified (OPTCD=ASYNC), the ECB-EXIT field of the DPL (DPLECBXR) is used in this manner:

ECB=*event_control_block_addr* DNR uses the field as the address of an external ECB. The application program is responsible for issuing a WAIT macro instruction and clearing the ECB when posted. If EXIT=*exit_routine_address* is specified, DNR uses the field as the address of an exit routine, and schedules the routine as indicated in the following EXIT operand.

ECB=INTERNAL, DNR uses the field as an internal ECB. The application program is responsible for issuing a WAIT macro instruction specifying the internal ECB, but need not clear it when posted.

Note: If synchronous mode is specified (OPTCD=SYNC), the parameter list is flagged for processing as if ECB=INTERNAL was specified, and the ECB-EXIT field is used as an internal ECB that is waited on and cleared automatically before returning control to the application program.

Default: INTERNAL (internal ECB).

EXIT=*exit_routine_address*

Address of a routine to schedule when the request completes.

The EXIT and ECB operands share the same storage location in the parameter list and are therefore mutually exclusive. The completion exit is scheduled only if asynchronous mode is specified by OPTCD=ASYNC. If synchronous mode was indicated, the exit routine is not used.

If one is specified with this operand, the address is overwritten with an internal ECB before the request completes.

Default: Not indicated (no exit routine).

MF = (I | L | G | M | E, [*dpl_address*])

Standard (inline), list, generate, modify, or execute form of the DIRSRV macro instruction.

The second sublist operand, *dpl_address*, specifies the address of the parameter list to use for this request. If no MF operand is specified, the standard form is used.

Note: It is recommended to use the list form of the macro to ensure DPL alignment on a full word boundary. If you do not use the list form, make sure you align the storage area used for the remote parameter list on a fullword boundary. This ensures that the internal ECB is aligned on a fullword boundary.

Default: MF=I (inline, nonreentrant).

Completion Information

The DIRSRV macro instruction completes normally (or conditionally) when the requested information is returned in the storage area provided by the application program. The length of the storage area is updated to reflect the actual amount of information returned.

On normal return to the application program, the general return code in register 15 is set to zero (DROKAY), and the conditional completion code is returned in register zero. The DPL return code field is set accordingly. DCALIAS is set if the name referenced in the NABUF and NALEN operands is not the official RPC name associated with the returned data. If DCALIAS is set and a storage area was supplied in the QNBUF and QNLEN operands, the DNR returns the official RPC name in the QNBUF storage area. If the official RPC name could not fit in the storage area, DCOVRFLO is set.

If the DIRSRV macro instruction completes abnormally, no information is returned in the storage area and the storage area length is unmodified. The general return code in register 15 and the recovery action code indicate the nature of the failure.

- If the general return code is set to DRFAILED, the recovery action code is returned in register zero and the DPL return code contains a specific error code that identifies a particular error
- If the general return code is set to DRFATLPTL, the recovery action code and the error code are both returned in register zero and the DPL is not updated

Return Codes

The following table lists the symbolic return codes for the GET-RPC-BYNAME macro. The values associated with the symbolic names can be found in the DPL macro expansion.

General Return Code	Recovery Action Code	Conditional or Specific Error Code		
DROKAY	DAOKAY	DCOKAY	DCALIS	DCOVRFLO
DRFAILED	DAEXCPTN	DENONAMEDENOTFND DENAMODE	DENOVALUD ENOCDS DEVAMODE	DENOQNAMDE OVRFLO DEQNMODE
	DAENVIRO	DESYSERR DENOTACT DEUNAVBL DETERM	DESUBSYS DENOTRDY DERSOURC	DENOTCNF DESTOP DENOTPRB
	DAFORMAT	DEBDOPCD DEBDEXIT	DEBDFNCD DEBDNAME	DEBDXECB DEBDVALU
DRFATLPTL	DAPROCED	DEACTIVE		
	DADPLERR	DEBDTYPE	DEPROTCT	DEPLMODE

Example

This example shows the use of GET-RPC-BYNAME. The request is to find the RPC number for the RPC name portmapper.

The application supplies this DIRSRV information:

NABUF = (address of:)

U	N	R	P	C

NALEN = 6
VABUF = (address of:)

VALEN = 100
QNBUF = (address of:)

QNLEN = 100
SIZE = 0

This information is returned:

NABUF = (address of:)

U	N	R	P	C

NALEN = 6
VABUF = (address of:)

0x01	0x86	0xa0

VALEN = 4
QNBUF = (address of:)

O	R	T	M	A	P	P	E	R

QNLEN = 10

Because the search string, SUNRPC is given as an alias in the RPC configuration file, the official RPC name is returned in the qualified name buffer.

GET-RPC-BYVALUE

The GET-RPC-BYVALUE instance of the DIRSRV macro instruction is used to return an RPC name when its number is known. The information returned is obtained locally from the RPC configuration member (DNRRPC*xx*).

```
[ symbol ] DIRSRV GET,RPC,BYVALUE,
                  NABUF = name_address,
                  NALEN = name_length,
                  VABUF = value_address,
                  VALEN = value_length
                  [ ,SYSID = MVS_subsystem_id ]
                  [ ,OPTCD = SYNC | ASYNC ]
                  [ ,ECB = INTERNAL | event_control_block_addr ]
                  [ ,EXIT = exit_routine_address ]
                  [ ,MF = ( I | L | G | M | E, [ dpl_address ] ) ]
```

GET,RPC,BYVALUE

DNR is to search for and return an RPC name associated with the RPC number provided by the application program.

NABUF = *name_address*

Address of a storage area in which the DNR returns an RPC name.

Default: Zero (no name storage area).

NALEN = *name_length*

Length (in bytes) of the storage area identified by the NABUF operand. The length is updated when the request completes reflecting the actual amount of information returned. If the request completes abnormally, no information is returned and the length remains unchanged.

Default: Zero (no name returned)

VABUF = *value_address*

Address of a storage area in which the application program has placed a four-byte RPC number.

Default: Zero (no value storage area).

VALEN = *value_length*

Length (in bytes) of the RPC number located in the storage area identified by the VABUF operand. The value of the storage area must be four (bytes).

Default: Zero (no value provided)

SYSID = *MVS_subsystem_id*

ID of the MVS subsystem that will process this request. Normally this operand is not required and an installation default is used.

The *MVS_subsystem_id* is an alphanumeric string up to four characters in length.

Note: If more than one DNR subsystem is active on the local system, the particular subsystem that will process this request must be specified.

Default: Not indicated (use installation default).

OPTCD = SYNC | ASYNC

Synchronization mode used when executing this macro instruction.

OPTCD=SYNC the request executes in synchronous mode, and control is not returned to the application program until the requested macro instruction completes.

OPTCD=ASYNC the request executes in asynchronous mode, and returns control to the application program immediately after it is accepted by DNR. The application program is notified asynchronously when the request completes. The ECB and EXIT operands determine the form of asynchronous notification.

Default: SYNC (synchronous mode).

ECB = INTERNAL | *event_control_block_addr*

Location of an event control block (ECB) posted by DNR when the directory request completes. The ECB can be any fullword of storage aligned on a fullword boundary.

The ECB and EXIT operands share the same storage location in the Directory Services Parameter List (DPL), and are therefore mutually exclusive operands. If asynchronous mode is specified (OPTCD=ASYNC), the ECB-EXIT field of the DPL (DPLECBXR) is used in this manner:

ECB=*event_control_block_addr* DNR uses the field as the address of an external ECB. The application program is responsible for issuing a WAIT macro instruction and clearing the ECB when posted. If EXIT=*exit_routine_address* is specified, DNR uses the field as the address of an exit routine, and schedules the routine as indicated in the following EXIT operand text.

ECB=INTERNAL DNR uses the field as an internal ECB. The application program is responsible for issuing a WAIT macro instruction specifying the internal ECB, but need not clear it when posted.

Note: If synchronous mode is specified (OPTCD=SYNC), the parameter list is flagged for processing as if ECB=INTERNAL was specified, and the ECB-EXIT field is used as an internal ECB that is waited on and cleared automatically before returning control to the application program.

Default: INTERNAL (internal ECB).

EXIT=*exit_routine_address*

Address of a routine to schedule when the request completes. The EXIT and ECB operands share the same storage location in the parameter list and are therefore mutually exclusive.

The completion exit is scheduled only if asynchronous mode is specified by OPTCD=ASYNC. If synchronous mode was specified, the exit routine is not used. If one is specified with this operand, the address is overwritten with an internal ECB before the request completes.

Default: Not indicated (no exit routine).

MF = (I | L | G | M | E, [*dpl_address*])

Standard (inline), list, generate, modify, or execute form of the DIRSRV macro instruction.

The second sublist operand, *dpl_address*, specifies the address of the parameter list to use for this request. If no MF operand is specified, the standard form is used.

Note: It is recommended you use the list form of the macro to ensure DPL alignment on a full word boundary. If you do not use the list form, make sure you align the storage area used for the remote parameter list on a fullword boundary. This ensures that the internal ECB is aligned on a fullword boundary.

Default: MF=I (inline, nonreentrant).

Completion Information

The DIRSRV macro instruction completes normally (or conditionally) when the requested information is returned in the storage area provided by the application program. The length of the storage area is updated to reflect the actual amount of information returned.

On normal return to the application program, the general return code in register 15 is set to zero (DROKAY), and the conditional completion code is returned in register zero. The DPL return code field is set accordingly.

If the DIRSRV macro instruction completes abnormally, no information is returned in the storage area and the storage area length is unmodified. The general return code in register 15 and the recovery action code indicate the nature of the failure.

- If the general return code is set to DRFAILED, the recovery action code is returned in register zero and the DPL return code contains a specific error code that identifies a particular error
- If the general return code is set to DRFATLPTL, the recovery action code and the error code are both returned in register zero and the DPL is not updated

Return Codes

The following table lists the symbolic return codes for the GET-RPC-BYVALUE macro. The values associated with the symbolic names can be found in the DPL macro expansion.

General Return Code	Recovery Action Code	Conditional or Specific Error Code		
DROKAY	DAOKAY	DCOKAY		
DRFAILED	DAEXCPTN	DENONAMEDENOCDS DEVAMODE	DENOVALU DEOVRFLO	DENOTFND DENAMODE
	DAENVIRO	DESYSERR DENOTACT DEUNAVBL DETERM	DESUBSYS DENOTRDY DERSOURC	DENOTCNF DESTOP DENOTPRB
	DAFORMAT	DEBDOPCD DEBDEXIT	DEBDFNCD DEBDNAME	DEBDXECB DEBDVALU
DRFATLPTL	DAPROCED	DEACTIVE		
	DADPLERR	DEBDTYPE	DEPROTCT	DEPLMODE

Example

This example shows the use of GET-RPC-BYVALUE. The request is to find the RPC name for the RPC value 100000.

The application supplies this DIRSRV information:

NABUF = (address of:)

NALEN = 100
VABUF = (address of:)

0x01 0x86 0xa0

VALEN = 4

This information is returned:

NABUF = (address of:)

O R T M A P P E R

NALEN = 10
VABUF = (address of:)

0x01 0x86 0xa0

VALEN = 4

PURGE

The PURGE instance of the DIRSRV macro instruction is used to purge a previously issued asynchronous DIRSRV request. The DNR searches for the DPL and completes the request.

[symbol] DIRSRV PURGE,

MF = (E, *dpl_address*])

PURGE

DNR is to purge a previously issued DIRSRV request.

MF =
(E, [*dpl_address*])

The execute form of the PURGE macro instruction.

The second sublist operand, *dp_ address*, specifies the address of the parameter list to purge.

Note: It is recommended you use the list form of the macro to ensure DPL alignment on a full word boundary. If you do not use the list form, make sure you align the storage area used for the remote parameter list on a fullword boundary. This ensures that the internal ECB is aligned on a fullword boundary.

Default: MF=E

Return Codes

The following table lists the symbolic return codes for the PURGE macro. The values associated with the symbolic names can be found in the DPL macro expansion.

General Return Code	Recovery Action Code	Conditional or Specific Error Code
DROKAY	DAOKAY	DCOKAY
DRFAILED	DAENVIRO	DESYSERR

Appendix

A

MF Operand Summary

This appendix includes the macro instruction forms indicated by the MF operand. It includes these sections:

- Macro Instruction Forms Supported by the API—Describes the macro instruction forms supported by the API
- MF Operands Supported by API Macro Instructions—Describes the MF operands supported by the API macro instructions
- Short, Long and Extended Parameter List Forms —Discusses the long and short parameter list forms
- Internal API Macro Instructions —Describes internal macro instructions for expanding TPL-based macro instructions

Macro Instruction Forms Supported by the API

Macro instruction forms are indicated by the MF operand. The first sublist operand is the form type. The second sublist operand, if coded, must be the address of the storage area that contains, or will contain, the parameter list associated with the macro instruction.

Macro Instruction Form	Description
MF=I	Generate parameter list in-line with macro instruction, as well as instructions in-line to execute parameter list.
MF=L	Generate parameter list in-line with macro instruction. Do not generate instructions to execute parameter list.
MF=(L,addr)	Generate instructions in-line with macro instruction to build parameter list in remote storage area indicated by addr. Do not generate instructions to execute parameter list.

Macro Instruction Form	Description
MF=(G,addr)	Generate instructions in-line with macro instruction to build parameter list in remote storage area indicated by addr, as well as instructions to execute parameter list.
MF=(M,addr)	Generate instructions in-line with macro instruction to modify parameter list in remote storage area indicated by addr. Do not generate instructions to execute parameter list.
MF=(E,addr)	Generate instructions in-line with macro instruction to modify parameter list in remote storage area indicated by addr, as well as generate instructions to execute parameter list.
MF=DSECT	Generate a dummy control section in-line with macro instruction that maps the fields of the control block.

Note: Macro instruction forms are discussed in detail in the chapter "Assembler Language Macro Instructions."

MF Operands Supported by API Macro Instructions

The following table indicates which macro instruction forms are supported for each API macro instruction.

When:

- A Yes appears in a column, it indicates the corresponding form is supported for the particular macro instruction
- Default appears in a column, it indicates the corresponding form is the default if the MF operand is not coded

Macro Instruction	Macro Instruction Forms						
	MF=I	**MF=L**	**MF=L,addr**	**MF=G,addr**	**MF=M,addr**	**MF=E,addr**	**MF=DSECT**
ACLOSE	MF operand not supported for this macro instruction						
AOPEN	MF operand not supported for this macro instruction						
APCB	No	Default	No	No	No	No	Yes
TACCEPT	Default	Yes	Yes	Yes	Yes	Yes	No
TADDR	Default	Yes	Yes	Yes	Yes	Yes	No
TBIND	Default	Yes	Yes	Yes	Yes	Yes	No
TCHECK	No	No	No	No	No	Default	No
TCLEAR	Default	Yes	Yes	Yes	Yes	Yes	No
TCLOSE	Default	Yes	Yes	Yes	Yes	Yes	No
TCONFIRM	Default	Yes	Yes	Yes	Yes	Yes	No
TCONNECT	Default	Yes	Yes	Yes	Yes	Yes	No
TDISCONN	Default	Yes	Yes	Yes	Yes	Yes	No

Macro Instruction	Macro Instruction Forms						
	MF=I	MF=L	MF=L,addr	MF=G,addr	MF=M,addr	MF=E,addr	MF=DSECT
TDSECT	MF operand not supported for this macro instruction						
TERROR	No	No	No	No	No	Default	No
TEXEC	Default	No	No	Yes	No	Yes	No
TEXLST	No	Default	Yes	No	Yes	No	No
TINFO	Default	Yes	Yes	Yes	Yes	Yes	No
TLISTEN	Default	Yes	Yes	Yes	Yes	Yes	No
TOPEN	Default	Yes	Yes	Yes	Yes	Yes	No
TOPTION	Default	Yes	Yes	Yes	Yes	Yes	No
TPL	No	Default	Yes	No	Yes	No	No
TRECV	Default	Yes	Yes	Yes	Yes	Yes	No
TRECVERR	Default	Yes	Yes	Yes	Yes	Yes	No
TRECVFR	Default	Yes	Yes	Yes	Yes	Yes	No
TREJECT	Default	Yes	Yes	Yes	Yes	Yes	No
TRELACK	Default	Yes	Yes	Yes	Yes	Yes	No
TRELEASE	Default	Yes	Yes	Yes	Yes	Yes	No
TRETRACT	Default	Yes	Yes	Yes	Yes	Yes	No
TSEND	Default	Yes	Yes	Yes	Yes	Yes	No
TSENDTO	Default	Yes	Yes	Yes	Yes	Yes	No
TSTATE	No	No	No	No	No	Default	No

Macro Instruction	Macro Instruction Forms						
	MF=I	MF=L	MF=L,addr	MF=G,addr	MF=M,addr	MF=E,addr	MF=DSECT
TUNBIND	Default	Yes	Yes	Yes	Yes	Yes	No
TUSER	Default	Yes	Yes	Yes	Yes	Yes	No

Short, Long and Extended Parameter List Forms

Most API macro instructions are TPL-based, indicating that they use a TPL for passing parameters to the requested transport service function and returning information to the application program. The TPL is normally 64 bytes in length and contains all the field necessary to store parameters for any API function. The same TPL can be used for all macro instructions as long as the appropriate operands are specified and fields are initialized as required.

However, many TPL-based macro instructions do not require a full-size TPL since only a subset of the information contained in the TPL is interpreted by the function executed. Therefore, the API supports a short form of the TPL for most TPL-based macro instructions. The TPL is organized so that the most frequently required fields occur at the beginning, and the least frequently used fields occur at the end.

Finally, the extended TPL adds a suffix to the standard length TPL. The suffix contains ALETs for all possible TPL parameters that address other data areas. Extended TPLs enable these data areas to reside in other address spaces.

Macro Instruction Rules

For a given macro instruction, the short form TPL is a contiguous subset of the long (standard) form. The size of this subset is function-specific.

These rules apply:

- The minimum size for a short form TPL is 20 bytes.

  If the macro instruction does not use any of the variable-length operand fields (for example, ADLEN and ADBUF) and does not pass or return any parameters other than option codes and return codes, the minimum size TPL can be used. This applies to macro instructions that use no operands other than FNCCD, OPTCD, EP, ECB, and EXIT and return no parameters other than the normal completion codes (that is, RTNCD).

- Macro instructions that do not use any of the variable-length operand fields but do pass or return one or more parameters in addition to the normal option codes and return codes, require 12 more bytes for a total of 32 bytes. Macro instructions that do not fall into the first group and do not use the ADLEN, ADBUF, DALEN, DABUF, OPLEN, or OPBUF operands can use a 32-byte TPL.

- The short form TPL size for the remaining macro instructions is determined on a case-by-case basis, depending on the number of variable-length parameters required for typical uses of the macro instruction.

Example

If only the protocol address parameter is required (that is, ADLEN and ADBUF operands), a 40-byte TPL can be used. If a protocol address or user data is required, a 48-byte TPL can be used. Generally, less frequently used parameters such as protocol options and user connect data are excluded for the purpose of determining the length of the short form TPL.

Internal API Macro Instructions

The API uses several internal macro instructions for expanding TPL-based macro instructions. These have been included in the macro library provided with the API.

The APIMZGBL Macro Instruction

The APIMZGBL macro instruction is used to set global constants that are referenced by other macro instructions. Since some of these constants affect the appearance of the assembler language listing, the APIMZGBL macro instruction is partially defined in this section.

Note: Only those operands that may be of interest to the application programmer are documented.

Assembler Format Description

This is the assembler format description for the APIMZGBL macro instruction:

[*symbol*] APIMZGBL [COMMENT=*comment column number*]

[,R0 = *symbol for register 0*]

[,R1 = *symbol for register 1*]

[,R13 = *symbol for register 13*]

[,R14 = *symbol for register 14*]

[,R15 = *symbol for register 15*

symbol	The symbolic name of the macro.
COMMENT = comment column number	Initial column number for comments in generated assembler language statements. The operand must be an integer value between 16 and 71, inclusive. Default: 36.

R0 = symbol for register 0

Symbol used to represent general register zero in generated assembler language statements.

If an alphanumeric symbol is specified, the symbol must be defined elsewhere in the input stream using an EQU statement.

Default: Zero (use standard numeric symbol).

R1 = symbol for register 1

Symbol used to represent general register one in generated assembler language statements.

If an alphanumeric symbol is specified, the symbol must be defined elsewhere in the input stream using an EQU statement.

Default: One (use standard numeric symbol).

R13 = symbol for register 13

Symbol used to represent general register 13 in generated assembler language statements.

If an alphanumeric symbol is specified, the symbol must be defined elsewhere in the input stream using an EQU statement.

Default: 13 (use standard numeric symbol).

R14 = symbol for register 14

Symbol used to represent general register 14 in generated assembler language statements.

If an alphanumeric symbol is specified, the symbol must be defined elsewhere in the input stream using an EQU statement.

Default: 14 (use standard numeric symbol).

R15 = symbol for register 15

Symbol used to represent general register 15 in generated assembler language statements.

If an alphanumeric symbol is specified, the symbol must be defined elsewhere in the input stream using an EQU statement.

Default: 15 (use standard numeric symbol).

Example

This example sets the comment column to 41, and redefines the general register symbols:

```
APIMZGBL COMMENT=41,R0=R0,R1=R1,R13=R13,R14=R14,R15=R15
```

Appendix B

Macro Instruction Operand Summary

This appendix summarizes the operands for the API macro instructions.

The tables in this appendix summarize the operands for all API macro instructions. For each macro instruction, all positional and keyword operands are listed. Positional operands are indicated by their position within the SYSLIST variable symbol, and keyword operands are indicated by their keyword name.

For example:

- SYSLIST(1) indicates the first positional operand
- SYSLIST(*n*) indicates the *n*th positional operand
- ADBUF indicates a keyword operand whose keyword name is ADBUF

Information Provided

The following information is provided for each operand:

- Operand position or keyword
- Operand format
- Default value
- DSECT label defining the parameter list location
- Operand description
- Operand Format

The operand format defines how the operand is generated and stored in the corresponding parameter list. For simple list forms of macro instructions (MF=L), the type of DC instruction that generates the operand at assembly time is listed. For all other forms, the instruction that expands to generate the operand at execution time is listed. If an entry is blank, either the operand cannot be specified with the indicated form, or the macro form is not supported for the particular macro instruction. N/A indicates that the operand does not cause a value to be stored in the corresponding parameter list, and is not applicable.

Integer Notes

For macro instructions that generate a parameter list, the default value generated when the operand is not specified is listed. An integer value enclosed in square brackets references one of these notes:

[1] The EXIT operand is mutually exclusive with the ECB operand.

If neither is specified, the default value for the ECB operand applies.

[2] The PROTO operand is mutually exclusive with the TYPE operand.

If neither is specified, the default value for the TYPE operand applies.

[3] For the TEXEC and TPL macro instructions, the FNCCD operand defaults to zero if not specified.

For all other TPL-based macro instructions, the function code parameter is generated in accordance with the macro instruction used to generate the parameter list.

If the operand specifies a value stored in a parameter list, the DSECT label corresponding to the operand is listed:

- The first label defines the location in the parameter list where the operand is stored.
- The second label following a colon (:) specifies the EQU used to define bit fields within location. Any label listed in parentheses is an alternate name for the same location.

Macro Instruction Operands

This section includes the operand summary for all macros, listed in alphabetical order.

ACLOSE Operand Format

	Operand Format		Default Value	DSECT Label	Description
Operand	MF=L	Other			
APCB		LA		N/A	APCB Address

AOPEN Operand Format

	Operand Format		Default Value	DSECT Label	Description
Operand	MF=L	Other			
APCB		LA		N/A	APCB Address

APCB Operand Format

	Operand Format		Default Value	DSECT Label	Description
Operand	MF=L	Other			
ACNTX	A		0	APCBACTX	Application Context
AM	AL1		TLI	APCBAM: APCBAMSK	Access Method
APPLID	CL8		0	APCBAPPL	Application Name
ECNTX	A		0	APCBECTX	Environment Context
ENVIRO	AL1		ASM	APCBENVR	Run-Time Environment
EXLST	A		0	APCBEXLS	Exit List
OPTCD	AL1		TRACE	APCBOPTC: APCBOTRC	Option Codes
PASSWD	CL8		0	APCBPSWD	Application Password

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
RMODE	AL1		ANY	APCBFLAG: APCBFANY	Residency Mode
SYSID	CL4		ACSS	APCBAMID	MVS Subsystem ID

TACCEPT Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
DABUF	A	LA	0	TPLDABUF	User Data Address
DALEN	A	LA	0	TPLDALEN	User Data Length
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
NEWEP		L	0	TPLNEWEP (TPLPARM2)	New Endpoint Identifier
OPBUF	A	LA	0	TPLOPBUF	Protocol Options Address
OPLEN	A	LA	0	TPLOPLEN	Protocol Options Length
OPTCD	AL1	NI-OI	SYNC LONG NONEGOT	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD1:TONEGOT	Option Codes
SEQNO		L	0	TPLSEQNO (TPLPARM1)	Sequence Number

TADDR Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
ADBUF	A	LA	0	TPLADBUF	Protocol Address Address
ADLEN	A	LA	0	TPLADLEN	Protocol Address Length
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPTCD	AL1	NI-OI	SYNC LONG NOTRUNC LOCAL	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD1:TOTRUNC TPLOPCD3:TOREMOTE	Option Codes

TBIND Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
ADBUF	A	LA	0	TPLADBUF	Protocol Address Address
ADLEN	A	LA	0	TPLADLEN	Protocol Address Length
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPTCD	AL1	NI-OI	SYNC LONG NOTRUNC NONEGOT USE	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD1:TOTRUNC TPLOPCD1:TONEGOT TPLOPCD3:TOASSIGN	Option Codes
QLSTN	A	LA	0	TPLQLSTN (TPLPARM1)	Listen Queue Size

Note: TCHECK macro instruction has no keyword operands other than MF and TPL.

TCLEAR Operand Format

	Operand Format		Default Value	DSECT Label	Description
Operand	MF=L	Other			
DABUF	A	LA	0	TPLDABUF	User Data Address
DALEN	A	LA	0	TPLDALEN	User Data Length
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPTCD	AL1	NI-OI	SYNC LONG NOTRUNC	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD1:TOTRUNC	Option Codes

TCLOSE Operand Format

	Operand Format		Default Value	DSECT Label	Description
Operand	MF=L	Other			
ASCB		L	0	TPLASCB (TPLPARM2)	ASCB Address
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPTCD	AL1	NI-OI	SYNC LONG DELETE	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD3:TOPASS	Option Codes
TCB		L	0	TPLTCB (TPLPARM1)	TCB Address

TCONFIRM Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
ADBUF	A	LA	0	TPLADBUF	Protocol Address Address
ADLEN	A	LA	0	TPLADLEN	Protocol Address Length
DABUF	A	LA	0	TPLDABUF	User Data Address
DALEN	A	LA	0	TPLDALEN	User Data Length
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	ECB Address
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPBUF	A	LA	0	TPLOPBUF	Protocol Options Address
OPLEN	A	LA	0	TPLOPLEN	Protocol Options Length
OPTCD	AL1	NI-OI	SYNC LONG NOTRUNC BLOCK	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD1:TOTRUNC TPLOPCD1:TONOBLOK	

TCONNECT Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
ADBUF	A	LA	0	TPLADBUF	Protocol Address Address
ADLEN	A	LA	0	TPLADLEN	Protocol Address Length
DABUF	A	LA	0	TPLDABUF	User Data Address
DALEN	A	LA	0	TPLDALEN	User Data Length
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPBUF	A	LA	0	TPLOPBUF	Protocol Options Address
OPLEN	A	LA	0	TPLOPLEN	Protocol Options Length
OPTCD	AL1	NI-OI	SYNC LONG NONEGOT	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD1:TONEGOT	Option Codes

TDISCONN Operand Format

	Operand Format		Default Value	DSECT Label	Description
Operand	MF=L	Other			
DABUF	A	LA	0	TPLDABUF	User Data Address
DALEN	A	LA	0	TPLDALEN	User Data Length
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPTCD	AL1	NI-OI	SYNC LONG CLEAR	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD2:TOABORT	Option Codes

TDSECT Operand Format

	Operand Format		Default Value	DSECT Label	Description
Operand	MF=L	Other			
SYSLIST(n)	N/A	N/A		N/A	Data Structure Name
DOMAIN	N/A	N/A	INET	N/A	Communications Domain

TERROR Operand Format

	Operand Format		Default Value	DSECT Label	Description
Operand	MF=L	Other			
SYSLIST(1)	N/A	N/A	SUMMARY	N/A	Message Format

Note: Macro instruction has no keyword operands other than MF and TPL

TEXEC Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
FNCCD	AL1	LA	[3]	TPLFNCCD	Function Code

Note: Any keyword operand valid for other TPL-based macro instructions (except TOPEN).

TEXLST Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
SYSLIST(1)	N/A	N/A	AOPEN	N/A	Exit List Type
CONFIRM	A	LA	0	TXLCONF	Exit Routine Address
CONNECT	A	LA	0	TXLCONN	Exit Routine Address
DATA	A	LA	0	TXLDATA	Exit Routine Address
DGERR	A	LA	0	TXLDGERR	Exit Routine Address
DISCONN	A	LA	0	TXLDISC	Exit Routine Address
LERAD	A	LA	0	TXLLERAD	Exit Routine Address
RELEASE	A	LA	0	TXLRELSE	Exit Routine Address
SYNAD	A	LA	0	TXLSYNAD	Exit Routine Address
TPEND	A	LA	0	TXLTPEND	Exit Routine Address
APEND	A	LA	0	TXLAPEND	Exit List Address Table

TINFO Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
DABUF	A	LA	0	TPLDABUF	User Data Address
DALEN	A	LA	0	TPLDALEN	User Data Length
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPTCD	AL1	NI-OI	SYNC LONG NOTRUNC PRIMARY	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD1:TOTRUNC TPLOPCD4:TOINFO	Option Codes

TLISTEN Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
ADBUF	A	LA	0	TPLADBUF	Protocol Address Address
ADLEN	A	LA	0	TPLADLEN	Protocol Address Length
DABUF	A	LA	0	TPLDABUF	User Data Address
DALEN	A	LA	0	TPLDALEN	User Data Length
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
OPBUF	A	LA	0	TPLOPBUF	Protocol Options Address
OPLEN	A	LA	0	TPLOPLEN	Protocol Options Length
OPTCD	AL1	NI-OI	SYNC LONG NOTRUNC BLOCK	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD1:TOTRUNC TPLOPCD1:TONOBLOK	Option Codes

TOPEN Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
APCB	A	LA	0	TPLAPCB	APCB Address
ASCB		L	0	TPLASCB (TPLPARM2)	ASCB Address
DOMAIN	AL1	LA	INET	TPLDOM	Communications Domain
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
EXLST	A	LA	0	TPLEXLST	Exit List Address

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
OPTCD	AL1	NI-OI	LONG SYNC TUB PLAIN NEW	TPLOPCD1:TOSHORT TPLOPteCD1:TOASYNC TPLOPCD3:TOACEE TPLOPCD3:TOCIPHER TPLOPCD3:TOOLD	Option Codes
PROTO	AL2	LA	[2]	TPLPROTO	Protocol Number
SVCID	CL8	MVC	0	TPLSVCID	Service Name
TCB		L	0	TPLTCB (TPLPARM1)	TCB Address
TYPE	AL2	LA	COTS	TPLTYPE	Service Type
UCNTX	A	LA	0	TPLUCNTX	User Context
USER	A	LA	0	TPLUSER (TPLPARM3)	TUB or ACEE Address Table

TOPTION Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPBUF	A	LA	0	TPLOPBUF	Protocol Options Address

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
OPLEN	A	LA	0	TPLOPLEN	Protocol Options Length
OPTCD	AL1	NI-OI	SYNC LONG NONEGOT DECLARE TP	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD1:TONEGOT TPLOPCD4:TOOPTION TPLOPCD4:TOAPI	Option Codes

TPL Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
FNCCD	AL1	LA	[3]	TPLFNCCD	Function Code

Note: Any keyword operand valid for other TPL-based macro instructions (except TOPEN).

TRECV Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
DABUF	A	LA	0	TPLDABUF	User Data Address
DALEN	A	LA	0	TPLDALEN	User Data Length
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPTCD	AL1	NI-OI	SYNC LONG BLOCK DIRECT	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD1:TONOBLOK TPLOPCD2:TOINDIR	Option Codes

TRECVERR Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
ADBUF	A	LA	0	TPLADBUF	Protocol Address Address
ADLEN	A	LA	0	TPLADLEN	Protocol Address Length
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPBUF	A	LA	0	TPLOPBUF	Protocol Options Address
OPLEN	A	LA	0	TPLOPLEN	Protocol Options Length
OPTCD	AL1	NI-OI	SYNC LONG NOTRUNC	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD1:TOTRUNC	Option Codes Table

TRECVFR Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
ADBUF	A	LA	0	TPLADBUF	Protocol Address Address
ADLEN	A	LA	0	TPLADLEN	Protocol Address Length
DABUF	A	LA	0	TPLDABUF	User Data Address
DALEN	A	LA	0	TPLDALEN	User Data Length
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPBUF	A	LA	0	TPLOPBUF	Protocol Options Address
OPLEN	A	LA	0	TPLOPLEN	Protocol Options Length
OPTCD	AL1	NI-OI	SYNC LONG NOTRUNC BLOCK DIRECT	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD1:TOTRUNC TPLOPCD1:TONOBLOK TPLOPCD2:TOINDIR	Option Codes

TREJECT Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
DABUF	A	LA	0	TPLDABUF	User Data Address
DALEN	A	LA	0	TPLDALEN	User Data Length
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPTCD	AL1	NI-OI	SYNC LONG CLEAR	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD2:TOABORT	Option Codes
SEQNO		L	0	TPLSEQNO (TPLPARM1)	Sequence Number

TRELACK Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPTCD	AL1	NI-OI	SYNC LONG BLOCK CLEAR	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD1:TONOBLOK TPLOPCD2:TOABORT	Option Codes

TRELEASE Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPTCD	AL1	NI-OI	SYNC LONG	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT	Option Codes

TRETRACT Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPTCD	AL1	NI-OI	SYNC LONG	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT	Option Codes Table

TSEND Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
DABUF	A	LA	0	TPLDABUF	User Data Address
DALEN	A	LA	0	TPLDALEN	User Data Length
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPTCD	AL1	NI-OI	SYNC LONG NOMORE NORMAL EOM DIRECT	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD2:TOMORE TPLOPCD2:TOEXPDTE TPLOPCD2:TONOTEOM TPLOPCD2:TOINDIR	Option Codes

TSENDTO Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
ADBUF	A	LA	0	TPLADBUF	Protocol Address Address
ADLEN	A	LA	0	TPLADLEN	Protocol Address Length
DABUF	A	LA	0	TPLDABUF	User Data Address
DALEN	A	LA	0	TPLDALEN	User Data Length
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPBUF	A	LA	0	TPLOPBUF	Protocol Options Address
OPLEN	A	LA	0	TPLOPLEN	Protocol Options Length
OPTCD	AL1	NI-OI	SYNC LONG NONEGOT DIRECT	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD1:TONEGOT TPLOPCD2:TOINDIR	Option Codes

Note: TSTATE Macro instruction has no keyword operands other than MF and TPL.

TUNBIND Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
ECB	A	LA	INTERNAL	TPLECB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPTCD	AL1	NI-OI	SYNC LONG	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT	Option Codes

TUSER Operand Format

Operand	Operand Format		Default Value	DSECT Label	Description
	MF=L	Other			
ECB	A	LA	INTERNAL	TPLCB (TPLECBXR)	ECB Address
EP		L	0	TPLEPID (TPLEP)	Endpoint Identifier
EXIT	A	LA	[1]	TPLEXIT (TPLECBXR)	Completion Exit Address
OPTCD	AL1	NI-OI	SYNC LONG TUB PLAIN	TPLOPCD1:TOASYNC TPLOPCD1:TOSHORT TPLOPCD3:TOACEE TPLOPCD3:TOCIPHER	Option Codes
USER	A	LA	0	TPLUSER (TPLPARM3)	TUB or ACEE Address Table

Appendix

C

Register Usage Summary

This appendix shows how the general-purpose registers are set when control is returned to the next sequential instruction following the execution of an API macro instruction.

The table in the following section indicates which registers are left unchanged by the macro instruction, and which ones may be modified between the time the macro instruction is executed and control is returned to the application program. The table also shows the disposition of registers when any exit routine receives control. For a detailed discussion of exit routines, refer to TCPaccess *Assembler API Concepts*.

The following are the notes used in the next section when an integer is enclosed in square brackets.

[1] Register 13 must contain the address of an 18-word save area when the macro instruction is executed.

The API uses this save area to save and restore registers 2-12 before returning to the application program.

[2] Register 15 contains a general return code that indicates the overall success or failure of the macro instruction.

Based on the contents of register 15, register zero is set as the following table indicates:

Register 15		Register 0
0	X'00'	Conditional completion code
4	X'04'	Recovery action code
8	X'08'	Function code
12	X'0C'	Diagnostic code
16	X'10'	Diagnostic code
20	X'14'	Diagnostic code

If an error occurrs and the SYNAD or LERAD exit routine was invoked, registers zero and 15 contain the values returned by the exit routine.

If an error occurred and no SYNAD or LERAD exit routine exits, the API sets register 15 to four (X'04') and returns a recovery action code in register zero. Fatal errors resulting in a general return code greater than four never invoke the SYNAD or LERAD exit routines.

If an error occurs during the execution of a TERROR or TSTATE macro instruction, the SYNAD and LERAD exit routines are bypassed, and control is returned directly to the application program.

[3] When the SYNAD or LERAD exit routine is invoked, registers 2-12 are restored to their original contents at the time the macro instruction causing the error was executed.

API Register Usage

Instance	Contents Of General-Purpose Registers					
	Register 0	Register 1	Register 2-12	Register 13	Register 14	Register 15
On return from AOPEN and ACLOSE macro instructions	Zero if successful, specific error code otherwise	Address of APCB	Unmodified	Unmodified [1]	Unpredictable	General return code
On return from TCHECK macro instruction	Conditional completion code if successful, otherwise see note [2]	Address of TPL	Unmodified	Unmodified [1]	Unpredictable	General return code [2]
On return from TERROR macro instruction	Address of TEM if successful, otherwise see note [2]	Address of TPL	Unmodified	Unmodified [1]	Unpredictable	General return code [2]
On return from TSTATE macro instruction	Endpoint state (TSW) if successful, otherwise see note [2]	Address of TPL	Unmodified	Unmodified [1]	Unpredictable	General return code [2]
On return from TEXLST macro instruction	Unmodified	Address of TXL	Unmodified	Unmodified	Unmodified	General return code (always zero)
On return from all other TPL-based macro instructions OPTCD=SYNC	Conditional completion code if successful, otherwise see note [2]	Address of TPL	Unmodified	Unmodified [1]	Unpredictable	General return code [2]
On return from all other TPL-based macro instructions OPTCD= ASYNC	Zero if accepted, otherwise see note [2]	Address of TPL	Unmodified	Unmodified [1]	Unpredictable	General return code [2]

Instance	Contents Of General-Purpose Registers					
	Register 0	Register 1	Register 2-12	Register 13	Register 14	Register 15
On invocation of a SYNAD or LERAD exit routine	Recover action code	Address of TPL	Unmodified [3]	Unmodified [1]	API return address	Address of exit routine
On invocation of a TPL completion exit routine	Unpredictable	Address of TPL	Unpredictable	Unpredictable	API return address	Address of exit routine
On invocation of all other asynchronous exit routines	Unpredictable	Address of TPL	Unpredictable	Unpredictable	API return address	Address of exit routine

Appendix

D

Data Structures (Assembler Language)

This appendix includes the data structures provided by the application program as arguments of TCPaccess transport service functions or generated by the API and referenced by the application program. It defines the API data structures as used by application programs written in assembler language. Refer to the TCPaccess *C/Socket API Programmer's Reference* for definitions of the same data structures as used by application programs written in C language. This appendix includes these sections:

- Generating Dummy Control Sections—Describes the standard and alternate methods for generating dummy control sections (DSECTs)
- Assembler Language Definitions—Includes the assembler language definition DSECT code as used by application programs written in assembler language.

Generating Dummy Control Sections

Most dummy control sections (DSECTs) listed in this appendix can be generated with the TDSECT macro instruction, described in the chapter "Assembler Language Macro Instructions."

Data Structure Names

The data structure name of each DSECT to be generated must be included in the macro instruction's operand list. For each API data structure listed in this table, a standard and alternate method for generating the DSECT is given.

Data Structure	Macro Instruction Required to Generate DSECT			
	Standard Method	Alternate Method		
APCB	[symbol] APCB MF=DSECT	[symbol]	APIDAPCB	MF=DSECT
APCBXL	[symbol] APCB MF=DSECT	[symbol]	APIDAPCB	MF=DSECT
TEM	[symbol] TDSECT TEM	TEM	APIDTEM	MF=DSECT
TIB	[symbol] TDSECT TIB	TIB	APIDTIB	MF=DSECT
TPA	[symbol] TDSECT TPA	TPA	APIDTPA	MF=DSECT
TPL	[symbol] TDSECT TPL	TPL	APIDTPL	MF=DSECT
TPO	[symbol] TDSECT TPO	TPO	APIDTPO	MF=DSECT
TSW	[symbol] TDSECT TSW	TSW	APIDTSW	MF=DSECT
TUB	[symbol] TDSECT TUB	TUB	APIDTUB	MF=DSECT
TXL	[symbol] TDSECT TXL	TXL	APIDTXL	MF=DSECT
TXP	[symbol] TDSECT TXP	TXP	APIDTXP	MF=DSECT

Assembler Language Definitions

This section includes the following assembler language definition DSECT code:

- Application Program Control Block (APCB)
- APCB exit list
- Transport Endpoint Error Message (TEM)
- Transport Service Information Block (TIB)
- Transport Protocol Address (TPA)
- Transport Service Parameter List (TPL)
- Transport Protocol Options (TPO)
- Transport Endpoint State Word (TSW)
- Transport Endpoint User Block (TUB)
- Transport Endpoint Exit List (TXL)
- Transport Endpoint Exit Parameters (TXP)

APCB (Application Program Control Block)

```
*THE FOLLOWING DSECT DEFINES THE STRUCTURE AND CONTENT OF
*THE APPLICATION PROGRAM CONTROL BLOCK (APCB). THE APCB
*CONTAINS INFORMATION ASSOCIATED WITH THE APPLICATION
*PROGRAM AND IS USED TO MAINTAIN A SESSION WITH THE API
*SUBSYSTEM. THE ADDRESS OF THE APCB IS INCLUDED IN THE
*OPERAND FIELD OF AN AOPEN OR ACLOSE MACRO INSTRUCTION.
APCB       DSECT      APPLICATION PROGRAM CONTROL BLOCK
APCBTAG    DS   CL4             CONTROL BLOCK ID
APCBSL     DS   F               CONTROL BLOCK LENGTH
APCBAM     DS   X               ACCESS METHOD AND VERSION
APCBAMSK   EQU  B'11110000'   ACCESS METHOD ID
APCBATLI   EQU  1               TRANSPORT LAYER INTERFACE
APCBAMAX   EQU  APCBATLI        MAXIMUM ACCESS METHOD ID
APCBAVER   EQU  B'00001111'   ACCESS METHOD VERSION
APCBFLAG   DS   X             FLAG BYTE
APCBFSTP   EQU  B'10000000'     APPLID IS STEPNAME FROM TIOT
APCBF31B   EQU  B'01000000'     AMODE=31
APCBFANY   EQU  B'00100000'     RMODE=ANY
APCBFOPN   EQU  B'00010000'     APCB IS OPEN
APCBFERR   EQU  B'00001000'     PERMANENT ERROR FLAG
APCBFTRM   EQU  B'00000100'     TASK TERMINATION IN PROGRESS
APCBFECB   EQU  B'00000010'     ECB EVENT NOTIFY BLOCK
APCBFBSY   EQU  B'00000001'     OPEN/CLOSE IN PROGRESS
APCBOPTC   DS   X             OPTION CODE
APCBOTRC   EQU  B'10000000'     OPTCD=NOTRACE|TRACE
APCBOGTF   EQU  B'01000000'     OPTCD=NOGTF|GTF
APCBABRT   EQU  B'00100000'     OPTCD=ABORT
APCBAUTH   EQU  B'00010000'     OPTCD=AUTHEXIT
*          EQU  B'00001000'     RESERVED
*          EQU  B'00000100'     RESERVED
*          EQU  B'00000010'     RESERVED
*          EQU  B'00000001'     RESERVED
           DS   X             RESERVED
APBCENVR   DS   X             LANGUAGE ENVIRONMENT CODE
```

```
APCBASM    EQU  0              ASSEMBLER LANGUAGE
APCBIBMC   EQU  1              IBM C
APCBSASC   EQU  2              SAS C
APCBPLI    EQU  3              PLI
APCBCOBL   EQU  4              COBOL
APCBFORT   EQU  5              FORTRAN
APCBEMAX   EQU  APCBFORT       MAXIMUM ENVIRONMENT CODE
APCBERRC   DS   X            ERROR CODE
APCBECFG   EQU  1              SUBSYSTEM NOT CONFIGURED
APCBEACT   EQU  2              SUBSYSTEM NOT ACTIVE
APCBERDY   EQU  3              SUBSYSTEM NOT INITIALIZED
APCBESTP   EQU  4              SUBSYSTEM IS STOPPING
APCBEDRA   EQU  5              SUBSYSTEM IS DRAINING
APCBEVCK   EQU  6              APCB VALIDITY CHECK ERROR
APCBELER   EQU  7              INTERNAL LOGIC ERROR
APCBEPRB   EQU  8              NOT ISSUED FROM PRB
APCBEOPN   EQU  9              APCB ALREADY OPENED
APCBECLS   EQU  10             APCB ALREADY CLOSED
APCBEBSY   EQU  11             APCB IS BUSY WITH AOPEN/ACLOSE
APCBEPER   EQU  12             APCB HAS PERMANENT ERROR
APCBECVT   EQU  13             ACCESS METHOD CVT NOT AVAILABLE
APCBEMEM   EQU  14             INSUFFICIENT MEMORY AVAILABLE
APCBEENV   EQU  15             CANNOT INITIAL./TERMIN. ENVIR.
APCBEBEG   EQU  16             CANNOT ESTABLISH API SESSION
APCBEVER   EQU  17             INVALID ACCESS METHOD VERSION
APCBEOPT   EQU  18             INVALID/UNSUPPORTED OPTION
APCBEDUP   EQU  19             DUPLICATE SESSION FOR THIS AM
APCBEAMD   EQU  20             AMODE INCONSISTENT WITH AOPEN
APCBETRV   EQU  21             AMTV VALIDITY CHECK ERROR
APCBEEND   EQU  22             CANNOT RELEASE API SESSION
APCBDGNC   DS   XL2          DIAGNOSTIC CODE
APCBAMCB   DS   A            ACCESS METHOD CONTROL BLOCK
APCBAMCV   DS   A            ACCESS METHOD COMMUNICATION VECTOR
APCBAMTV   DS   A            ACCESS METHOD UAS TRANSFER VECTOR
APCBAMID   DS   CL4          ACCESS METHOD SUBSYSTEM ID
           DS   F            RESERVED
APCBEXLS   DS   A            APPLIC.-LEVEL EXIT LIST ADDRESS
APCBACTX   DS   F            APPLIC.-LEVEL CONTEXT VARIABLE
APCBECTX   DS   F            ENVIRO.-LEVEL CONTEXT VARIABLE
APCBAPPL   DS   CL8          APPLICATION ID
APCBPSWD   DS   CL8          APPLICATION PASSWORD
APCBLEN    EQU  *-APCB       LENGTH OF APCB
```

APCBXL (APCB Exit List)

```
*THE FOLLOWING DSECT DEFINES THE AM-INDEPENDENT STRUCTURE
*OF AN APCB EXIT LIST. IT IS NECESSARY TO DEFINE THIS
*STRUCTURE AT THIS LEVEL SO AOPEN CAN VALIDITY CHECK THE
*EXIT LIST.
APCBXL     DSECT            COMMON EXIT LIST FORMAT
APCBXLEN   DS    F          TOTAL LENGTH OF EXIT LIST
APCBXLST   DS    0A         LIST OF EXIT ROUTINE ENTRY POINTS
```

TEM (Transport Endpoint Error Message)

```
*THE FOLLOWING DSECT DEFINES THE STRUCTURE AND CONTENT OF
*AN ERROR MESSAGE RETURNED BY THE TERROR MACRO INSTRUCTION.
*THE INFORMATION RETURNED IS FORMATTED AS A MULTI-LINE WTO
*PARAMETER LIST (WTO MF=L), AND CAN BE SUPPLIED DIRECTLY
*TO A WTO MF=E MACRO INSTRUCTION. THE APPLICATION PROGRAM
*MAY USE THIS DSECT TO MANIPULATE CERTAIN FIELDS WITHIN THE
*PARAMETER LIST.
TEM       DSECT             TRANSPORT ENDPOINT ERROR MESSAGE
TEMTAG    DS   CL4          CONTROL BLOCK IDENTIFIER
TEMSL     DS   F            SUBPOOL AND LENGTH
TEMWTO    DS   0F           WTO PARAMETER LIST
TEMSGLEN  DS   AL2          MESSAGE LENGTH + 4 (FIRST LINE)
TEMMCSF1  DS   X            MCS FLAG BYTE #1
TEMMCSF2  DS   X            MCS FLAG BYTE #2
TEMSGTXT  DS   CL34         MESSAGE TEXT (FIRST LINE)
          ORG  TEMSGTXT
TEMSGID   DS   CL8            MESSAGE ID
TEMSGBDY  DS   CL26           MESSAGE BODY
          ORG
TEMDESC   DS   BL.16        DESCRIPTOR CODES
TEMROUT   DS   BL.16        ROUTING CODES
TEMSGTYP  DS   XL2          MLWTO LINE TYPE (FIRST LINE)
TEMAREA   DS   X            MLWTO AREA ID
TEMNLINE  DS   AL1          MLWTO NUMBER OF LINES
TEMMLLEN  DS   AL2          MLWTO LINE LENGTH + 4
TEMMLTYP  DS   XL2          MLWTO LINE TYPE
TEMMLTXT  DS   (*-*)C       MLWTO LINE TEXT
TEMLEN    EQU  *-TEM        MINIMUM LENGTH OF MULTI-LINE TEM
```

TIB (Transport Service Information Block)

```
*THE FOLLOWING DSECT DEFINES THE STRUCTURE AND CONTENT OF
*TRANSPORT INFORMATION RETURNED BY THE TINFO (OPTCD=PRIMARY)
*TRANSPORT SERVICE FUNCTION. THE FORMAT OF THIS INFORMATION
*IS STANDARD FOR ALL TRANSPORT PROVIDERS, AND IS INTENDED TO
*CONVEY THE BASIC CHARACTERISTICS OF THE UNDERLYING PROTOCOL
*AND SERVICE.
TIB       DSECT             TRANSPORT SERVICE INFORMATION BLOCK
TIBTSDOM  DS   X            TRANSPORT SERVICE DOMAIN
TIBDINET  EQU  2              DARPA INTERNET
TIBDACP   EQU  4              ACP ONLY API
TIBTSTYP  DS   X            TRANSPORT SERVICE TYPE
TIBTCOTS  EQU  1              CONNECTION-MODE SERVICE
TIBTCLTS  EQU  2              CONNECTIONLESS-MODE SERVICE
TIBTRAW   EQU  3              RAW MODE SERVICE
TIBTSCHR  DS   X              TRANSPORT SERVICE CHARACTERISTICS
*         EQU  B'10000000'      RESERVED
*         EQU  B'01000000'      RESERVED
*         EQU  B'00100000'      RESERVED
TIBCTSDU  EQU  B'00010000'      TSDU BOUNDARIES PRESERVED
TIBCXPDT  EQU  B'00001000'      EXPEDITED DATA SUPPORTED
TIBCOPTN  EQU  B'00000100'      USER-SETTABLE OPTS SUPPORTED
TIBCCOND  EQU  B'00000010'      CONNECT USER DATA SUPPORTED
TIBCDISD  EQU  B'00000001'      DISCONNECT USER DATA SUPPORTED
TIBTSOPT  DS   X              TRANSPORT SERVICE OPTIONS
TIBOASSO  EQU  B'10000000'      CLTS ASSOCIATIONS SUPPORTED
TIBOSCND  EQU  B'01000000'      SECONDARY INFO. AVAILABLE
TIBOSTAT  EQU  B'00100000'      STATISTICAL INFO. AVAILABLE
TIBORLSE  EQU  B'00010000'      COTS ORDERLY RELEASE SUPPORTED
*         EQU  B'00001000'      RESERVED
*         EQU  B'00000100'      RESERVED
*         EQU  B'00000010'      RESERVED
```

```
*         EQU  B'00000001'      RESERVED
TIBSYSID  DS   CL4            TRANSPORT PROVIDER'S SUBSYS. NAME
TIBSVCID  DS   CL8            TRANSPORT PROVIDER'S SERVICE NAME
TIBPROTO  DS   F              TRANSPORT PROTOCOL NUMBER
TIBQLSTN  DS   F              MAXIMUM SIZE OF LISTEN QUEUE
TIBQSEND  DS   F              MAXIMUM SIZE OF SEND QUEUE
TIBQRECV  DS   F              MAXIMUM SIZE OF RECEIVE QUEUE
TIBLTSND  DS   F              MAXIMUM SIZE OF TSEND/TSENDTO DATA
TIBLTRCV  DS   F              MAXIMUM SIZE OF TRECV/TRECVFR DATA
TIBLSEND  DS   F              MAXIMUM SIZE OF SEND BUFFER
TIBLRECV  DS   F              MAXIMUM SIZE OF RECEIVE BUFFER
TIBLADDR  DS   F              MAXIMUM SIZE OF PROTOCOL ADDRESS
TIBLOPTN  DS   F              MAXIMUM SIZE OF PROTOCOL OPTIONS
TIBLTSDU  DS   F              MAXIMUM SIZE OF TRANSP. DATA UNIT
TIBLXPDT  DS   F              MAXIMUM SIZE OF EXPED. DATA UNIT
TIBLCONN  DS   F              MAXIMUM SIZE OF CONNECT DATA
TIBLDISC  DS   F              MAXIMUM SIZE OF DISCONNECT DATA
TIBLINFO  DS   F              MAXIMUM SIZE OF INFORMATION UNIT
TIBLEN    EQU  *-TIB          LENGTH OF TIB
```

TPA (Transport Protocol Address)

```
*THE FOLLOWING DSECT DEFINES THE STRUCTURE AND CONTENT
*OF A TRANSPORT PROTOCOL ADDRESS IN THE INTERNET DOMAIN.
*THIS INFORMATION IS TRANSPORT PROVIDER-DEPENDENT AND
*IS NOT INTERPRETED BY API
*
*THE GENERAL FORMAT OF A TRANSPORT PROTOCOL ADDRESS IS:
*
*<DOMAIN><T-ADDR><N-ADDR>
*
*WHERE:
*
*DOMAIN = COMMUNICATION DOMAIN (SEE TOPEN)
*T-ADDR = TRANSPORT LAYER ADDRESS OF ENDPOINT
*N-ADDR = NETWORK LAYER ADDRESS OF TRANSPORT PROVIDER
TPA       DSECT             TRANSPORT PROTOCOL ADDRESS
TPAINET   DS   XL8          DOMAIN=INET TPA
          ORG  TPAINET
TPAINETD  DS   XL2          INTERNET DOMAIN
TPAINETT  DS   XL2          TCP PORT NUMBER
TPAINETN  DS   XL4          IP HOST ADDRESS
          ORG
LTPAINET  EQU  *-TPAINET     LENGTH OF INET TPA
```

TPL (Transport Service Parameter List)

```
*THE FOLLOWING DSECT DEFINES THE STRUCTURE OF THE TRANSPORT
*SERVICE PARAMETER LIST (TPL) FOR ALL TRANSPORT SERVICE
*FUNCTIONS. THE TPL BEGINS WITH A COMMON SECTION (I.E., THE
*PARAMETER LIST PREFIX) WHICH IS PRESENT FOR ALL FUNCTIONS.
TPL        DSECT          TRANSPORT SERVICE PARAMETER LIST
TPLIDENT   DS   X         CONTROL BLOCK ID
TPLIDSTD   EQU  234         STANDARD (LONG) FORMAT ID
TPLIDSHT   EQU  235         SHORT FORMAT ID
TPLIDEXT   EQU  236         EXTENDED FORMAT ID
TPLFNCCD   DSX             FUNCTION CODE
TFORG1     EQU  0           ORIGIN FOR STANDARD FUNCTIONS
TFACCEPT   EQU  1           ACCEPT CONNECTION REQUEST
TFADDR     EQU  2           ENABLE CONNECTIONS
TFBIND     EQU  3           BIND PROTOCOL ADDRESS
TFCLEAR    EQU  4           ACKNOWLEDGE DISCONNECT
TFCLOSE    EQU  5           CLOSE ENDPOINT
TFCONFRM   EQU  6           CONFIRM CONNECTION REQUEST
TFCONNCT   EQU  7           INITIATE CONNECTION REQUEST
TFDISCON   EQU  8           INITIATE DISCONNECT
TFINFO     EQU  9           GET TRANSPORT INFORMATION
TFLISTEN   EQU  10          LISTEN FOR CONNECTION REQUESTS
TFOPEN     EQU  11          OPEN ENDPOINT
TFOPTION   EQU  12          NEGOTIATE OPTIONS
TFRECV     EQU  13          RECEIVE FROM CONNECTION
TFRECVER   EQU  14          RECEIVE DATAGRAM ERROR
TFRECVFR   EQU  15          RECEIVE DATAGRAM
TFREJECT   EQU  16          REJECT CONNECTION REQUEST
TFRELACK   EQU  17          ACKNOWLEDGE CONNECTION RELEASE
TFRELESE   EQU  18          INITIATE CONNECTION RELEASE
TFRETRCT   EQU  19          RETRACT PENDING LISTEN
TFSEND     EQU  20          SEND TO CONNECTION
TFSENDTO   EQU  21          SEND DATAGRAM
TFUNBIND   EQU  22          UNBIND PROTOCOL ADDRESS
TFUSER     EQU  23          ASSOCIATE USER ID
TFMAX1     EQU  TFUSER      MAXIMUM FUNCTION ID
TFORG2     EQU  127         ORIGIN FOR CONTROL FUNCTIONS
TFERRORV   EQU  128         FORMAT ERROR MESSAGE VERBATIM
TFCHECK    EQU  129         CHECK TPL FOR COMPLETION
TFERROR    EQU  130         FORMAT ERROR MESSAGE
TFSTATE    EQU  131         GET ENDPOINT STATE
TFMAX2     EQU  TFSTATE     MAXIMUM FUNCTION ID
TPLACTIV   DS   X          SEMAPHORE (TPL ACTIVE)
TPLFLAGS   DS   X          FLAG BYTE
TPLFCMPL   EQU  B'10000000'    TPL COMPLETED
TPLFCERR   EQU  B'01000000'    COMPLETED WITH ERROR
TPLFXECB   EQU  B'00100000'    TPLECBXR IS EXTERNAL ECB
TPLFEXIT   EQU  B'00010000'    TPLECBXR IS EXIT ROUTINE
TPLF31B    EQU  B'00001000'    REQUEST ISSUED WITH AMODE=31
TPLFACPT   EQU  B'00000100'    ACCEPTING CONNECT INDICATION
*          EQU  B'00000010'    RESERVED
*          EQU  B'00000001'    RESERVED
TPLEP      DS   F           ENDPOINT
           ORG*-4
TPLEPID    DS   0F             ENDPOINT ID
TPLTCEP    DS   0A             TCEP ADDRESS
           ORG*+4
TPLECBXR   DS   A           IECB/XECB/EXIT
           ORG*-4
TPLECB     DS   0F          ECB PARAMETER
TPLIECB    DS   0F             INTERNAL ECB
TPLXECB    DS   0A             EXTERNAL ECB ADDRESS
TPLEXIT    DS  0A           EXIT ROUTINE ADDRESS
           ORG*+4
```

```
TPLOPTCD   DS   F           OPTION CODES
           ORG  TPLOPTCD
TPLOPCD1   DS   X             OPTION CODE #1
TOASYNC    EQU  B'10000000'     OPTCD=SYNC|ASYNC
TOSHORT    EQU  B'01000000'     OPTCD=LONG|SHORT
TOTRUNC    EQU  B'00100000'     OPTCD=NOTRUNC|TRUNC
TONEGOT    EQU  B'00010000'     OPTCD=NONEGOT|NEGOT
TOMBUF     EQU  B'00001000'     OPTCD=NOMBUF|MBUF
TONOBLOK   EQU  B'00000100'     OPTCD=BLOCK|NOBLOCK
TOEXTEND   EQU  B'00000010'     OPTCD=EXTEND
*          EQU  B'00000001'     RESERVED
TPLOPCD2   DS   X             OPTION CODE #2
TOMORE     EQU  B'10000000'     OPTCD=NOMORE|MORE
TOEXPDTE   EQU  B'01000000'     OPTCD=NORMAL|EXPEDITE
TONOTEOM   EQU  B'00100000'     OPTCD=EOM|NOTEOM
TOABORT    EQU  B'00010000'     OPTCD=CLEAR|ABORT
TOINDIR    EQU  B'00001000'     OPTCD=DIRECT|INDIR
*          EQU  B'00000100'     RESERVED
TOFULL     EQU  B'00000010'     OPTCD=NOFULL|FULL
TOTIME     EQU  B'00000001'     OPTCD=NOTIME|TIME
TPLOPCD3   DS   X             OPTION CODE #3
TOACEE     EQU  B'10000000'     OPTCD=TUB|ACEE
TOCIPHER   EQU  B'01000000'     OPTCD=PLAIN|CIPHER
TOOLD      EQU  B'00100000'     OPTCD=NEW|OLD
TOASSIGN   EQU  B'00010000'     OPTCD=USE|ASSIGN
TOREMOTE   EQU  B'00001000'     OPTCD=LOCAL|REMOTE
TOPASS     EQU  B'00000100'     OPTCD=DELETE|PASS
*          EQU  B'00000010'     RESERVED
*          EQU  B'00000001'     RESERVED
TPLOPCD4   DS   X             OPTION CODE #4
TOINFO     EQU  B'11000000'     TINFO OPTION CODES
TOPRIMRY   EQU  0                 OPTCD=PRIMARY
TOSCNDRY   EQU  1                 OPTCD=SECNDRY
TOSTATS    EQU  2                 OPTCD=STATS
TOOPTION   EQU  B'00110000'     TOPTION OPTION CODES
TODECLAR   EQU  0                 OPTCD=DECLARE
TOVERIFY   EQU  1                 OPTCD=VERIFY
TOQUERY    EQU  2                 OPTCD=QUERY
TODFAULT   EQU  3                 OPTCD=DEFAULT
TOAPI      EQU  B'00001000'       OPTCD=TP|API
*          EQU  B'00000100'       RESERVED
*          EQU  B'00000010'       RESERVED
*          EQU  B'00000001'       RESERVED
TPLRTNCD   DS   F               COMPOSITE RETURN CODE
           ORG  TPLRTNCD
TPLACTCD   DS   X                 RECOVERY ACTION CODE
TAOKAY     EQU  0                   SUCCESSFUL COMPLETION
TAEXCPTN   EQU  4                   EXCEPTIONAL CONDITION
TAINTEG    EQU  8                   CONNECTION/DATA INTEG. ERROR
TAENVIRO   EQU  12                  ENVIRONMENTAL CONDITION
TAFORMAT   EQU  16                  FORMAT OR SPECIF. ERRORS
TAPROCED   EQU  20                  SEQUENCE AND PROCED. ERRORS
TATPLERR   EQU  24                  LOGIC ERRORS W/NO TPL RTNCD
TAUSER     EQU  28                  USER-DEFINED ACTION CODES
TPLERRCD   DS   X                 SPECIFIC ERROR CODE
TCOKAY     EQU  B'00000000'         00: NO CONDITIONALS
TCVERIFY   EQU  B'10000000'         00: OPTIONS DID NOT VERIFY
TCNEGOT    EQU  B'01000000'         00: OPTIONS NEGOTIATED
TCTRUNC    EQU  B'00100000'         00: BUFFER TRUNCATED
TCSTOP     EQU  B'00001000'         00: SUBSYSTEM STOPPING
TCTIME     EQU  B'00000100'         00: TIMEOUT EXPIRED
TENONEGO   EQU  6                   04: NO NEGOTIATION ALLOWED
TENOBLOK   EQU  9                   04: NO BLOCKING ALLOWED
TENOLSTN   EQU  10                  04: NO LISTEN PENDING
TEPROTO    EQU  1                   08: PROTOCOL ERROR
TEOVRFLO   EQU  2                   08: BUFFER OVERFLOW
```

```
TEDISCON  EQU  3                  08: DISCONNECT RECEIVED
TERELESE  EQU  4                  08: ORDERLY RELEASE RCVD.
TEOVLAY   EQU  5                  08: CONTROL BLOCK OVERLAID
TEFLOW    EQU  9                  08: TEMPORARY FLOW CONTROL
TERETRCT  EQU  10                 08: LISTEN RETRACTED
TEPURGED  EQU  11                 08: PURGED BY TCLOSE
TESYSERR  EQU  1                  12: SYSTEM ERROR
TESUBSYS  EQU  2                  12: SUBSYSTEM ERROR
TENOTCNF  EQU  3                  12: SUBSYS NOT INSTALLED
TENOTACT  EQU  4                  12: SUBSYS NOT ACTIVE
TENOTRDY  EQU  5                  12: SUBSYS NOT INITIALIZED
TEDRAIN   EQU  6                  12: SUBSYS DRAINED BY OPER.
TESTOP    EQU  7                  12: SUBSYS STOPPED BY OPER.
TETERM    EQU  8                  12: SUBSYS ABNORMALLY TERM.
TEUNSUPO  EQU  9                  12: UNSUPPORTED OPT./FACIL.
TEUNSUPF  EQU  10                 12: UNSUPPORTED FUNC./SVC.
TEUNAVBL  EQU  11                 12: UNAVAILABLE SVC./FACIL.
TEUNAUTH  EQU  12                 12: USER UNAUTHORIZED
TERSOURC  EQU  13                 12: INSUFFICIENT RESOURCES
TEINUSE   EQU  14                 12: PROTOCOL ADDRESS IS ALREADY ENABLED
TEUSRXIT  EQU  15                 12: FAILED BY USER EXIT
TEBDOPCD  EQU  1                  16: INVLD OPTION CODE
TEBDEPID  EQU  2                  16: INVLD ENDPOINT ID
TEBDXECB  EQU  3                  16: INVLD ECB/EXIT ADDR.
TEBDDOM   EQU  4                  16: INVLD COMM. DOMAIN
TEBDPROT  EQU  5                  16: INVLD TRANSPRT PROTO.
TEBDTYPE  EQU  6                  16: INVLD TRANSPRT SVC TYPE
TEBDXLST  EQU  7                  16: INVLD EXIT LIST
TEBDUSER  EQU  8                  16: INVLD USER PARM
TEBDACEE  EQU  9                  16: INVLD ACCESSOR ELEMENT
TEBDSQNO  EQU  10                 16: INVLD SEQUENCE NUMBER
TEBDQLEN  EQU  11                 16: INVLD QUEUE LENGTH
TEBDTCB   EQU  12                 16: INVLD TCB ADDRESS
TEBDASCB  EQU  13                 16: INVLD ASCB ADDRESS
TEBDADDR  EQU  14                 16: INVLD PROTOCOL ADDRESS
TEBDOPTN  EQU  15                 16: INVLD OPTIONS
TEBDDATA  EQU  16                 16: INVLD DATA BUFFER
TEBDTSID  EQU  18                 16: INVLD TRANSPORT SVC. ID
TESTATE   EQU  1                  20: INVLD STATE FOR FUNC.
TEINEXIT  EQU  2                  20: INVLD FUNC. WITHIN EXIT
TEINACTV  EQU  3                  20: CHECK ISSUED TO INACT. TPL
TEINCMPL  EQU  4                  20: ENDPOINT HAS INCOMPL. FNC
TEINDICA  EQU  5                  20: PENDING CONNECT INDICA.
TEBUFOVR  EQU  6                  20: SEND/RCV. BUFFER OVERRUN
TEREQOVR  EQU  7                  20: SEND/RCV. RQST. OVERRUN
TENOCONN  EQU  8                  20: NO CONNECT INDICATION
TENODISC  EQU  9                  20: NO DISCONNECT INDICA.
TEOUTSEQ  EQU  10                 20: REQUEST OUT OF SEQUENCE
TENOERR   EQU  11                 20: NO ERROR INDICATION
TEAMODE   EQU  13                 20: AMODE CONFLICTS W/ APCB
TEOWNER   EQU  14                 20: NOT OPENED BY THIS TASK
TELISTEN  EQU  15                 20: LISTEN QUEUE FULL
TEACCEPT  EQU  16                 20: ACCEPTING TO ENDPOINT
TEB4EXIT  EQU  1                  24:TPL CHECKED BEFORE EXIT
TEACTIVE  EQU  2                  24:TPL IS STILL ACTIVE
TPLDGNCD  DS   H                DIAGNOSTIC AND SENSE CODES
TPLMIN    EQU  *-TPL           MINIMUM TPL LENGTH
TLRELACK  EQU  *-TPL           LENGTH OF SHORT TPL: TRELACK
TLRELESE  EQU  *-TPL           LENGTH OF SHORT TPL: TRELEASE
TLRETRCT  EQU  *-TPL           LENGTH OF SHORT TPL: TRETRACT
TLUNBIND  EQU  *-TPL           LENGTH OF SHORT TPL: TUNBIND
TPLPARM   DS   XL(4*3)         FIXED-LENGTH FUNCTION PARAMETERS
          ORG  TPLPARM
TPLPARM1  DS   F                 PARAMETER #1
          ORG  *-4
TPLQLSTN  DS   0F                  LISTEN QUEUE LENGTH
```

```
TPLSEQNO  DS   0F               SEQUENCE NUMBER
TPLTCB    DS   0A               TCB ADDRESS
TPLMBUFO  DS   0A               OPTCD=MBUF MBUF OFFSET
          ORG  *+4
TPLPARM2  DS   F              PARAMETER #2
          ORG  *-4
TPLNEWEP  DS   0F               NEW ENDPOINT
TPLASCB   DS   0A               ASCB ADDRESS
TPLCOUNT  DS   0F               RESIDUAL BYTE COUNT
          ORG  *+4
TPLPARM3  DS   F              PARAMETER #3
          ORG  *-4
TPLUSER   DS   0A               TUB OR ACEE ADDRESS
TPLDISCD  DS   0F               DISCONNECT REASON CODE
TPLDGERR  DS   0F               DATAGRAM ERROR CODE
TPLSTATE  DS   0F               OLD ENDPOINT STATE
TPLXCNT   DS   0F               XDATA RESIDUAL COUNT
          ORG  *+4
TLACCEPT  EQU  *-TPL      LENGTH OF SHORT TPL: TACCEPT
TLCLEAR   EQU  *-TPL      LENGTH OF SHORT TPL: TCLEAR
TLCLOSE   EQU  *-TPL      LENGTH OF SHORT TPL: TCLOSE
TLDISCON  EQU  *-TPL      LENGTH OF SHORT TPL: TDISCON
TLREJECT  EQU  *-TPL      LENGTH OF SHORT TPL: TREJECT
TLUSER    EQU  *-TPL      LENGTH OF SHORT TPL: TUSER
TPLVAPAR  DS   XL(8*3)    VARIABLE-LENGTH FUNCTION PARMS
          ORG  TPLVAPAR
TPLADDR   DS   XL(4*2)      PROTOCOL ADDRESS PARAMETER
          ORG  TPLADDR
TPLADBUF  DS   A              PARAMETER ADDRESS
TPLADLEN  DS   F              PARAMETER LENGTH
TLADDR    EQU  *-TPL      LENGTH OF SHORT TPL: TADDR
TLBIND    EQU  *-TPL      LENGTH OF SHORT TPL: TBIND
TLCONFRM  EQU  *-TPL      LENGTH OF SHORT TPL: TCONFIRM
TLCONNCT  EQU  *-TPL      LENGTH OF SHORT TPL: TCONNECT
TLLISTEN  EQU  *-TPL      LENGTH OF SHORT TPL: TLISTEN
TLRECVER  EQU  *-TPL      LENGTH OF SHORT TPL: TRECVERR
TPLDATA   DS   XL(4*2)      USER DATA PARAMETER
          ORG  TPLDATA
TPLDABUF  DS   A              PARAMETER ADDRESS
TPLDALEN  DS   F              PARAMETER LENGTH
TLINFO    EQU  *-TPL      LENGTH OF SHORT TPL: TINFO
TLRECV    EQU  *-TPL      LENGTH OF SHORT TPL: TRECV
TLRECVFR  EQU  *-TPL      LENGTH OF SHORT TPL: TRECVFR
TLSEND    EQU  *-TPL      LENGTH OF SHORT TPL: TSEND
TLSENDTO  EQU  *-TPL      LENGTH OF SHORT TPL: TSENDTO
TPLOPTN   DS   XL(4*2)      OPTIONS PARAMETER
          ORG  TPLOPTN
TPLOPBUF  DS   A              PARAMETER ADDRESS
TPLOPLEN  DS   F              PARAMETER LENGTH
TLOPTION  EQU  *-TPL      LENGTH OF SHORT TPL: TOPTION
          ORG  TPLLEN
          EQU  *-TPL      STANDARD (LONG) TPL LENGTH
*THE TPL FORMAT FOR TOPEN DIFFERS FROM THAT USED BY THE
*OTHER TRANSPORT SERVICE FUNCTIONS. IN PARTICULAR, FUNCTION
*ARGUMENTS UNIQUE TO TOPEN OVERLAY THE VARIABLE-LENGTH
*PARAMETER SECTION AS DEFINED BELOW.
          ORG  TPLVAPAR
TPLDOM    DS   X           COMMUNICATION DOMAIN
TDINETO   EQU  1           DARPA INTERNET PRE 3.1
TDINET    EQU  2           DARPA INTERNET
TDACP     EQU  4           ACP TASK GROUP DOMAIN
TDMAX     EQU  TDACP       MAXIMUM VALUE FOR DOMAIN
TPLOFLAG  DS   X           OPEN FLAGS/ENVIRONMENT
TPLOFPRO  EQU  B'10000000'   PROTOCOL NUMBER SPECIFIED
TPLOFORD  EQU  B'01000000'   COTS ORDERLY RELEASE REQUIRED
TPLOFASO  EQU  B'00100000'   CLTS ASSOCIATIONS REQUIRED
```

```
TPLOFECB  EQU  B'00010000'   EVENT LIST SPECIFIED
TPLOFSOC  EQU  B'00001000'   MODE=SOCKETS
*         EQU  B'00000100'   RESERVED
*         EQU  B'00000010'   RESERVED
*         EQU  B'00000001'   RESERVED
TPLSERVC  DS   H          TRANSPORT SERVICE REQUESTED
          ORG  *-2
TPLTYPE   DS   0H           TRANSPORT SERVICE TYPE
TTCOTS    EQU  1              CONNECTION-MODE SERVICE
TTCLTS    EQU  2              CONNECTIONLESS-MODE SVC.
TTRAW     EQU  3              RAW-MODE SERVICE
TTMAX     EQU  TTRAW          MAX. VALUE FOR SERVICE TYPE
TPLPROTO  DS   0H           TRANSPORT PROTOCOL NUMBER
          ORG  *+2
TPLAPCB   DS   A          APCB ADDRESS FOR API SESSION
TPLSVCID  DS   CL8        PROVIDER'S SERVICE NAME
TPLEXLST  DS   A          ADDRESS OF EXIT LIST
TPLUCNTX  DS   F          ONE WORD OF USER CONTEXT
TLOPEN    EQU  *-TPL      LENGTH OF SHORT TPL: TOPEN
          ORG
TPLMAX    EQU  *-TPL       MAXIMUM TPL LENGTH
*GENERAL RETURN CODES (RETURNED IN R15) ARE USED TO INDICATE
*SUCCESSFUL OR UNSUCCESSFUL COMPLETION OF A FUNCTION IN
*SYNCHRONOUS MODE, AND ACCEPTANCE OR NON-ACCEPTANCE OF A
*FUNCTION IN ASYNCHRONOUS MODE.
TROKAY    EQU  0           SUCCESSFUL COMPLETION/ACCEPTED
TRFAILED  EQU  4           UNSUCCESS. COMPLETION/NOT ACCPT.
TRFATLFC  EQU  8           INVALID FUNCTION CODE
TRFATLPL  EQU  12          FATAL TPL ERROR
TRFATLAM  EQU  16          FATAL ACCESS METHOD ERROR
TRFATLAP  EQU  20          APCB IS CLOSED
TRUSER    EQU  24          FIRST USER RETURN CODE
*
*THE DISCONNECT REASON CODE RETURNED INTPLDISCD
*AND DATAGRAM ERROR CODE RETURNED IN TPLDGERR ARE
*PROVIDER-DEPENDENT.
*
*
*THE FOLLOWING DISCONNECT REASON CODES MAY BE
*RETURNED BY THE API FOR INTERNET DOMAIN
*(DOMAIN=INET)
*
TDTRANTO  EQU  1           TRANSMISSION TIMEOUT
TDHOSTUN  EQU  2           HOST UNREACHABLE
TDPORTUN  EQU  3           PORT UNREACHABLE
TDRABORT  EQU  4           REMOTE ABORT
TDLNIDWN  EQU  5           LOCAL NETWORK I/F DOWN
TDPROTUN  EQU  6           PROTOCOL UNREACHABLE
TDACPRR   EQU  7           ACP CONNECTION ERROR
TDAPIRR   EQU  8           API CONNECTION ERROR
TDNETUN   EQU  9           NET UNREACHABLE
TDNOFRAG  EQU  10          FRAGMENTATION NEEDED/DF
TDSRFAIL  EQU  11          SOURCE ROUTE FAILED
```

```
*
*TPL EXTENSION
*
TPLPRM1X  DS   F          TPLPARM1 - ALET OR EXTENSION
TPLPRM2X  DS   F          TPLPARM2 - ALET OR EXTENSION
TPLPRM3X  DS   F          TPLPARM3 - ALET OR EXTENSION
          ORG  TPLPARM3X
TPLUSALT  DS   F          USER - ALET
          ORG  ,
TPLADALT  DS   F          ADBUF - ALET
TPLDAALT  DS   F          DAFUF - ALET
TPLOPALT  DS   F          OPBUF - ALET
TPLXDIAG  DS   F          EXTENDED DIAGNOSTIC CODE
TPLEXLEN  EQU  *          LENGTH OF EXTENDED TPL
```

TPO (Transport Protocol Options)

```
AM=TLI TRANSPORT PROTOCOL OPTIONS (TPO)
*****************************************************************
*-- TPO --
*
*AM=TLI TRANSPORT PROTOCOL OPTIONS
*****************************************************************
*THE FOLLOWING DSECT DEFINES THE STRUCTURE FOR SPECIFYING
*OPTIONS ASSOCIATED WITH A TRANSPORT SERVICE FUNCTION.
*GENERALLY SUCH OPTIONS WILL BE INDICATED WITH THE OPLEN
*AND OPBUF OPERANDS OF A TRANSPORT SERVICE MACRO INSTRUCTION.
*
*MORE THAN ONE OPTION MAY BE SPECIFIED IN A SINGLE MACRO
*INSTRUCTION WHERE EACH OPTION IS FORMATTED IN ACCORDANCE
*WITH THE FOLLOWING DSECT. OPBUF POINTS TO THE FIRST OPTION
*IN THE LIST, AND OPLEN IS THE TOTAL LENGTH OF THE OPTION
*LIST.
TPO       DSECT                TRANSPORT PROTOCOL OPTIONS
TPOPTLEN  DS   H               LENGTH OF THIS OPTION
TPOPTION  DS   H               OPTION NAME (I.E., NUMBER)
*
*     API-SPECIFIC OPTIONS
*
TPOAQSND  EQU  0                MAXIMUM NUMBER OF SENDS
TPOAQRCV  EQU  1                MAXIMUM NUMBER OF RECEIVES
TPOALSND  EQU  2                LENGTH OF SEND BUFFER
TPOALRCV  EQU  3                LENGTH OF RECEIVE BUFFER
TPOAMAX   EQU  TPOALRLN         MAXIMUM OPTION NUMBER
*
*     ACP-PROVIDER-SPECIFIC OPTIONS
*
TPOPRWND  EQU  1                TCP RECEIVE WINDOW
TPOPKTIM  EQU  2                KEEPALIVE TIME
TPOPKEEP  EQU  3                KEEPALIVE OPTIONS
TPOPDNAG  EQU  4                DEFEAT NAGLE ALGORITHM
TPOPRWND  EQU  1                TCP RECEIVE WINDOW
TPOPKTIM  EQU  2                KEEPALIVE TIME
TPOPKEEP  EQU  3                KEEPALIVE OPTIONS
TPOPDNAG  EQU  4                DEFEAT NAGLE ALGORITHM
TPOPRTIM  EQU  5                FULL RECEIVE TIMEOUT
TPOIPOPT  EQU  6                IP OPTIONS
TPOSIOAR  EQU  7                SOCKET ADD ROUTE
TPOSIODR  EQU  8                SOCKET DELETE ROUTE
TPOSIFCF  EQU  9                SOCKET INTERFACE CONFIG
TPOSIFLG  EQU  10               SOCKET INTERFACE FLAGS
TPOSIFMT  EQU  11               SOCKET INTERFACE MTU
TPOSIFME  EQU  12               SOCKET INTERFACE METRIC
TPOSIFNM  EQU  13               SOCKET IFC NETWORK MASK
```

```
TPOSIFBA  EQU  14              SOCKET BROADCAST ADDR
TPOSIFAD  EQU  15              SOCKET IFC ADDRESS
TPOSIFEN  EQU  16              SOCKET IFC ENET ADDRESS
TPOSIFNO  EQU  17              NUMBER OF INTERFACES
TPOSIFDS  EQU  18              DESTINATION ADDRESS
TPOIPTTL  EQU  19              IP TIME TO LIVE
TPOIPTOS  EQU  20              IP TYPE OF SERVICE
TPOTPMSS  EQU  21              TCP MAXIMUM SEGMENT SIZE
TPOIPDNR  EQU  22              IP DO NOT ROUTE
TPOIPBRO  EQU  23              IP BROADCAST
TPOUDSUM  EQU  24              UDP CHECKSUMS
TPOTQSND  EQU  25              MAXIMUM NUMBER OF SENDS
TPOTQRCV  EQU  26              MAXIMUM NUMBER OF RECEIVES
TPOTLSND  EQU  27              LENGTH OF SEND BUFFER
TPOTLRCV  EQU  28              LENGTH OF RECEIVE BUFFER
TPOREUSE  EQU  29              REUSE ADDRESS
TPOIMIF   EQU  30              SAW_IP_MULTICAST_IF (UNSUPPORTED)
TPOIMTTL  EQU  31              SAW_IP_MULTICAST_TTL (UNSUPPORTED)
TPOIMLOO  EQU  32              SAW_IP_MULTICAST_LOOP (UNSUPPORTED)
TPOIADDM  EQU  33              SAW_IP_ADD_MEMBERSHIP (UNSUPPORTED)
TPOIDRPM  EQU  34              SAW_IP_DROP_MEMBERSHIP (UNSUPPORTED)
TPOUDATA  EQU  35              User Data
TPOACCON  EQU  36              SO_ACCEPTCONN option
TPORCVLW  EQU  37              SO_RCVLOWAT option
TPOSNDLW  EQU  38              SO_SNDLOWAT option
TPOIPHDR  EQU  39              IP_HDRINCL option
TPOFIONR  EQU  40              FIONREAD option
TPOTMAX   EQU  TPOFIONR        MAXIMUM OPTION NUMBER
TPOVALUE  DS   (*-*)X          OPTION VALUE
```

TSW – Transport Endpoint State Word

```
*THE FOLLOWING DSECT DEFINES THE STRUCTURE AND CONTENT
*OF THE STATE WORD RETURNED BY THE TSTATE FUNCTION. THE
*STATE WORD CONTAINS STATUS FLAGS REPRESENTING PENDING
*ACTIVITY ON THE ENDPOINT, AND A HALFWORD STATE VALUE
*WHICH REPRESENTS THE STATE OF THE ENDPOINT AT THE MOST
*RECENT SUCCESSFUL COMPLETION OF A TRANSPORT SERVICE
*FUNCTION.
TSW       DSECT              TRANSPORT ENDPOINT STATE WORD
TSWSTATF  DC   X'00'          STATUS FLAGS
TSWFCHNG  EQU  B'10000000'      STATE IS CHANGING
TSWFACPT  EQU  B'01000000'      ACCEPTING TO THIS ENDPOINT
TSWOPNO   EQU  B'00100000'      OPENING OLD TO THIS EP
*         EQU  B'00010000'      RESERVED
*         EQU  B'00001000'      RESERVED
*         EQU  B'00000100'      RESERVED
*         EQU  B'00000010'      RESERVED
*         EQU  B'00000001'      RESERVED
TSWPENDF  DC   X'00'          PENDING FUNCTION INDICATORS
TSWPFCLS  EQU  B'10000000'      TCLOSE DELETE
TSWPFDIS  EQU  B'01000000'      TDISCONN, TCLEAR, TRETRACT
TSWPFREL  EQU  B'00100000'      TRELEASE
TSWPFACK  EQU  B'00010000'      TRELACK
TSWPFCON  EQU  B'00001000'      CONNECTION ESTABLISHMENT FNC
TSWPFLCL  EQU  B'00000100'      LOCAL ENDPOINT MANAGEMENT FNC
TSWPFPAS  EQU  B'00000010'      TCLOSE PASS
TSWPFOPN  EQU  B'00000001'      TOPEN
TSWPFRCV  EQU  B'00000000'      TRECV
TSWPFSND  EQU  B'00000000'      TSEND
TSWPFDGM  EQU  B'00000000'      TRECVFR, TSENDTO, TRECVERR
TSWSTATE  DC   H'0'           CURRENT ENDPOINT STATE
TSCLOSED  EQU  0                CLOSED (NON-EXISTENT)
TSOPENED  EQU  1                OPENED (NOT BOUND)
```

```
TSDSABLD  EQU  2              DISABLED (BOUND, QLSTN EQ 0)
TSENABLD  EQU  3              ENABLED (BOUND, QLSTN GT 0)
TSINCONN  EQU  4              CONNECT INDICATION PENDING
TSOUCONN  EQU  5              CONNECT IN PROGRESS
TSCONNCT  EQU  6              CONNECTED (OR ASSOCIATED)
TSINRLSE  EQU  7              RELEASE INDICATION PENDING
TSOURLSE  EQU  8              RELEASE IN PROGRESS
TSMAX     EQU  TSOURLSE       MAXIMUM STATE VALUE
TSWLEN    EQU  *-TSW          LENGTH OF TSW
```

TUB (Transport Endpoint User Block)

```
*THE FOLLOWING DSECT DEFINES THE STRUCTURE AND CONTENT OF
*USER ID PARAMETERS REQUIRED FOR ASSOCIATING A USER WITH
*AN ENDPOINT. THIS STRUCTURE SHOULD BE PROVIDED AS AN
*ARGUMENT TO TOPEN AND TUSER WHEN AN ACCESSOR ENVIRONMENT
*ELEMENT (ACEE) IS NOT AVAILABLE OR APPROPRIATE.
TUB       DSECT               TRANSPORT ENDPOINT USER BLOCK
TUBUID    DS   XL9              USER ID
          ORG  TUBUID
TUBUIDL   DS   X                  USER ID LENGTH
TUBUIDC   DS   CL8                USER ID CHARACTER STRING
          ORG
TUBGRP    DS   XL9              GROUP NAME
          ORG  TUBGRP
TUBGRPL   DS   X                  GROUP NAME LENGTH
TUBGRPC   DS   CL8                GROUP NAME CHARACTER STRING
          ORG
TUBPWD    DS   XL9              PASSWORD
          ORG  TUBPWD
TUBPWDL   DS   X                  PASSWORD LENGTH
TUBPWDC   DS   CL8                PASSWORD CHARACTER STRING
          ORG
TUBLEN    EQU  *-TUB            LENGTH OF TUB
```

TXL (Transport Endpoint Exit Lis)

```
*THE FOLLOWING DSECT DEFINES THE STRUCTURE AND CONTENT OF
*THE EXIT LIST PROVIDED BY THE APPLICATION PROGRAM AS AN
*ARGUMENT OF THE TOPEN AND AOPEN MACRO INSTRUCTIONS. EACH
*ENTRY IS THE ADDRESS OF AN EXIT ROUTINE WHICH IS TO RECEIVE
*CONTROL WHEN A PARTICULAR EVENT OCCURS. IF THE VALUE OF
*AN ENTRY IS ZERO, THE CORRESPONDING EXIT ROUTINE IS NOT
*DEFINED. THE EXIT LIST CAN BE GENERATED BY A TEXLST MACRO
*INSTRUCTION.
TXL         DSECT               TRANSPORT ENDPOINT EXIT LIST
TXLLENXL    DS   F              LENGTH OF EXIT LIST
TXLEXITS    DS   0A             LIST OF EXIT ROUTINES
TXLPROTO    DS   10A              PROTOCOL EVENT EXITS
            ORG  TXLPROTO
TXLCONN     DS   A                  CONNECT INDICATION
TXLCONF     DS   A                  CONFIRM INDICATION
TXLDATA     DS   A                  NORMAL DATA INDICATION
TXLXDATA    DS   A                  EXPEDITED DATA INDICATION
TXLDGERR    DS   A                  DATAGRAM ERROR INDICATION
TXLDISC     DS   A                  DISCONNECT INDICATION
TXLRELSE    DS   A                  RELEASE INDICATION
TXLSWIND    DS   A                  WINDOW OPEN INDICATION
            DS   2A                 RESERVED
            ORG
TXLTPEND    DS   A                  PROVIDER END EXIT
            DS   2A                 RESERVED
            DS   H                  RESERVED
TXLFECB     DS   H                  EVENT is ECB Flags
TXLFECI     EQU  X'8000'            Connect Ind. event is ECB
TXLFECF     EQU  X'4000'            Confirm event is ECB
TXLFEDA     EQU  X'2000'            Data Ind. event is ECB
TXLFEXD     EQU  X'1000'            Expedited data event is ECB
TXLFEDE     EQU  X'0800'            Datagram error event is ECB
TXLFEDI     EQU  X'0400'            Disconnect event is ECB
TXLFERL     EQU  X'0200'            Release Ind. event is ECB
TXLFESW     EQU  X'0100'            Send window opened is ECB
TXLFETP     EQU  X'0080'            TPEND event is an ECB
TXLLENTO    EQU  *-TXL          LENGTH OF TOPEN EXIT LIST
*THE FOLLOWING EXITS CAN ONLY BE SPECIFIED IN AN AOPEN
*EXIT LIST.
TXLERROR    DS   2A               SYNCHRONOUS ERROR EXITS
            ORG  TXLERROR
TXLSYNAD    DS   A                  PHYSICAL ERRORS
TXLLERAD    DS   A                  LOGICAL ERRORS
            ORG
TXLAPEND    DS   A                SYNCHRONOUS END EXIT
            DS   2A               RESERVED
TXLLENAO    EQU  *-TXL          LENGTH OF AOPEN EXIT LIST
TXLLEN      EQU  *-TXL          MAXIMUM LENGTH OF TXL
```

TXP (Transport Endpoint Exit Parameters)

```
*THE FOLLOWING DSECT DEFINES THE STRUCTURE AND CONTENT OF
*THE PARAMETER LIST WHOSE ADDRESS IS PASSED IN R1 TO THE
*TPEND AND ALL PROTOCOL EVENT EXIT ROUTINES. A TPL ADDRESS
*IS PASSED IN R1 TO THE SYNAD, LERAD, AND TPL COMPLETION
*EXIT ROUTINES.
TXP       DSECT            TRANSPORT ENDPOINT EXIT PARAMETERS
TXPTYPE   DS   H           EXIT TYPE
TXPTPROT  EQU  1             PROTOCOL EVENT EXIT
TXPTCMPL  EQU  2             ENDPOINT COMPLETION EXIT
TXPTPEND  EQU  3             PROVIDER END EXIT
TXPTSYNC  EQU  4             SYNCHRONOUS ERROR EXIT
TXPAPEND  EQU  5             SUBSYSTEM END EXIT
          DS   H           RESERVED
TXPEP     DS   F           ENDPOINT
          ORG  *-4
TXPEPID   DS   0F          ENDPOINT ID
TXPTCEP   DS   0A          TCEP ADDRESS
TXPAPCB   DS   0A          APCB ADDRESS FOR APEND
          ORG  *+4
TXPEXIT   DS   A           EXIT ROUTINE ENTRY POINT
TXPPARM   DS   F           EXIT PARAMETER
          ORG  *-4
TXPEVENT  DS   0F            PROTOCOL EVENT CODE
TXPECONN  EQU  0               CONNECT INDICATION
TXPECONF  EQU  4               CONFIRM INDICATION
TXPEDATA  EQU  8               NORMAL DATA INDICATION
TXPEXPDT  EQU  12              EXPEDITED DATA INDICATION
TXPERROR  EQU  16              DATAGRAM ERROR INDICATION
TXPEDISC  EQU  20              DISCONNECT INDICATION
TXPERLSE  EQU  24              ORDERLY RELEASE INDICATION
TXPESWND  EQU  28              SEND WINDOW OPEN
TXPREASN  DS   0F            TPEND REASON CODE
TXPRDRAN  EQU  0               OPERATOR DRAINED SUBSYSTEM
TXPRSTOP  EQU  4               OPERATOR STOPPED SUBSYSTEM
TXPRTERM  EQU  8               SUBSYS. ABNORM. TERMINATED
TXPTPL    DS   0A            COMPLETION TPL ADDRESS
          ORG  *+4
TXPACNTX  DS   F           APPLICATION-LEVEL CONTEXT
TXPUCNTX  DS   F           USER-LEVEL (ENDPOINT) CONTEXT
TXPECNTX  DS   F           ENVIRONMENT-LEVEL CONTEXT
TXPLEN    EQU  *-TXP       LENGTH OF TXP
TXPPARM2  DS   F              SECOND PARAMETER WORD
TXPCOUNT  EQU  TXPPARM2,4      DATA/SEND WINDOW COUNT
```

Index

A

B

C

D

E

F

G

I

K

L

M

N

O

P

R

S

T

U

Computer Associates™